Asymptotically optimal path planning with integrated local optimisation for robotic vine pruning

Scott Paulin

A thesis presented for the degree of
Doctor of Philosophy
in
Mechanical Engineering
at the
University of Canterbury,
Christchurch, New Zealand.

28 February 2017

ABSTRACT

Grapes are an important crop to New Zealand and the rest of the world. Vineyards cover 75,000km^2 worldwide and New Zealand wine exports were valued at $1.57 billion in 2016. Grape vines are pruned annually to improve yield. Pruning is done manually for many vine species and is the most labour intensive and expensive task for the vineyard.

A robotic pruning system is being developed at the University of Canterbury to automate cane pruning, where canes are selectively removed to leave a small number of long healthy canes. The robot uses stereo-cameras to construct a 3D model of the vines and an AI system to determine where vines should be pruned. A UR5 robot arm with a spinning router is then used to make the cuts. An online path planning algorithm is required to plan collision free paths for the robot arm to reach cut-points. The path planner should quickly compute paths that are fast to execute after being converted to trajectories so that the robot can operate efficiently. This thesis proposes new path planning approaches for quickly computing paths that are fast to execute when converted to trajectories.

Sampling based path planners are by far the most widely used path planning algorithms for high degree of freedom robot arms. These planners explore the robot's configuration space to find collision free path for the robot to follow. Some of these planners also attempt to optimise a solution as time permits. The performance of sampling-based path planners can be limited by the efficiency of the collision detector that they use, their (sometimes slow) convergence rate and how they can exploit a robot arm's redundancy to find high quality paths. Efficient operation of the vine pruning

robot relies on its path planner being able to quickly find paths that can be converted to fast-to-execute trajectories.

Collision detection speed is often the bottleneck in performance for many sampling based path planners. Fast path planning can be achieved by using an efficient collision detector. This research proposes a new collision detection algorithm that exploits the structure of grape vines to provide fast collision detection. This collision detector takes 3.0×10^{-6} seconds on average to classify the collision status of a robot configuration and is 50 times faster than the popular Flexible Collision Library [Pan et al 2012]. This speed-up in collision detection results in a 27 times reduction in path planning times.

Two approaches to finding fast-to-execute paths are to use an asymptotically optimal path planner, or to use a local optimiser. Asymptotically optimal planners are guaranteed to eventually find the optimal, e.g. shortest, solution but can be slow. Local optimisers are capable of quickly improving a solution with respect to a cost function such as path length, but are not guaranteed to find the optimal solution. The approach proposed in this thesis integrates an asymptotically optimal planner with a local optimiser to speed up the search for short paths while retaining the planner's asymptotic optimality. The asymptotically optimal RRTConnect* planner integrated with a 'short cut' local optimiser found paths that were 31% faster to execute than those found by RRTConnect* without the local optimiser for the vine pruning robot given three seconds of planning time.

Many robot arms are redundant with respect to their tasks. The robot arm might be able to accomplish the task, e.g. move the end-effector to a specific Cartesian position, using more than one set of joint angles. Ideally the robot's path planner would be able to use the extra configurations to find higher quality paths, however, little work has been done to investigate this. In this thesis these extra goal configurations are used to find significantly shorter paths that are faster to execute compared to a planner that chooses one goal configuration arbitrarily. A planner using these redundant goal configurations found paths that had 58% lower execution times on average compared to a planner that did not use the redundant goal configurations for the vine pruning

robot.

This thesis investigates ways to reduce the computation time and improve the solution quality of sampling based path planners, specifically for the task of pruning grape vines using a robot arm. Fast computation times are achieved using a new collision detector that exploits the structure of grape vines, and by integrating an asymptotically optimal path planner with a local optimiser. The robot arm's redundancy is exploited to allow the planner to discover fast-to-execute paths.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my supervisors Professor XiaoQi Chen, Associate Professor Richard Green and Dr. Tom Botterill for the advice and feedback they have provided me with during my studies. In particular, I would like to thank Tom for his advice, critical feedback, technical discussions and patience, which have all greatly contributed to my PhD experience and the skills that I take forward.

Finally I would like to thank my friends and family for supporting me during the PhD roller-coaster.

Chapter 1

INTRODUCTION

Grapes are an important crop to New Zealand and the rest of the world. Vineyards cover 75,000km^2 worldwide [FAO 2017] and New Zealand wine exports were valued at $1.57 billion in 2016 [NZ Winegrowers 2016]. Grape vines are pruned annually during the winter to improve yield and prevent disease [Kilby 1999]. Many vines in New Zealand are *cane pruned*, where branches are selectively cut to leave two or three long healthy canes [Christensen 2000]. Pruning is done manually for many vine species and is the most labour intensive and expensive task for the vineyard [Dryden 2014]. It is also one of the few tasks on the vineyard that has not been mechanised.

A robot is being developed at the University of Canterbury to automate cane pruning [Botterill et al 2016] (Fig. 1.1). This robot uses three stereo-cameras to construct a 3D model of the vines [Botterill et al 2013] and an AI system to find ideal places on the vine to prune [Corbett davies et al 2012] (Fig. 1.2). A UR5 robot arm with a spinning router is then used to make the cuts (Fig.1.3). A path planning algorithm (also referred to as a motion planning algorithm) is required to plan collision free configuration space (also referred to as joint space) paths for the robot arm to reach cut-points. The path is then converted to a trajectory by computing the times that each configuration should be reached by the robot arm. These collision free paths are used to guide the robot arm into positions to cut parts of the vine without collisions between the robot arm and itself, or the robot arm and the plant. The path planner should quickly compute fast-to-execute paths so that the robot can operate efficiently. This thesis proposes new path planning approaches for quickly computing fast-to-execute paths. This the-

sis proposes new path planning approaches that are tested on this vine pruning robot. The robot is fully described in Chapter 3.

A number of other robots have been developed to perform labour intensive agricultural tasks such as kiwi-fruit harvesting [Scarfe et al 2009], apple harvesting [Nguyen et al 2013, Davidson et al 2016], precision weeding [Lee et al 2014], and melon harvesting [Edan 1995, Edan et al 2000] among others [Hayashi et al 2010, Chatzimichali et al 2009, Liu et al 2012, Cai Jianrong, Wang Feng, LüQiang 2009, Noguchi and Terao 1997]. Like the vine pruning robot, all of these robots perform their tasks (harvesting or weeding) while stationary, with the exception of the melon harvesting robot [Edan et al 2000].

In their review of 50 recent agricultural robots, Bac et al [2014] found that sensing the environment and robot cycle times were two important challenges. The robot's cycle time is directly affected by the computation time and quality, e.g. shortness, of collision free paths found by the path planner. Imperfect sensing results in the path planner having a model of the world with errors. If these errors are not accounted for, the robot may collide with obstacles while following a path that the planner has computed to be collision free.

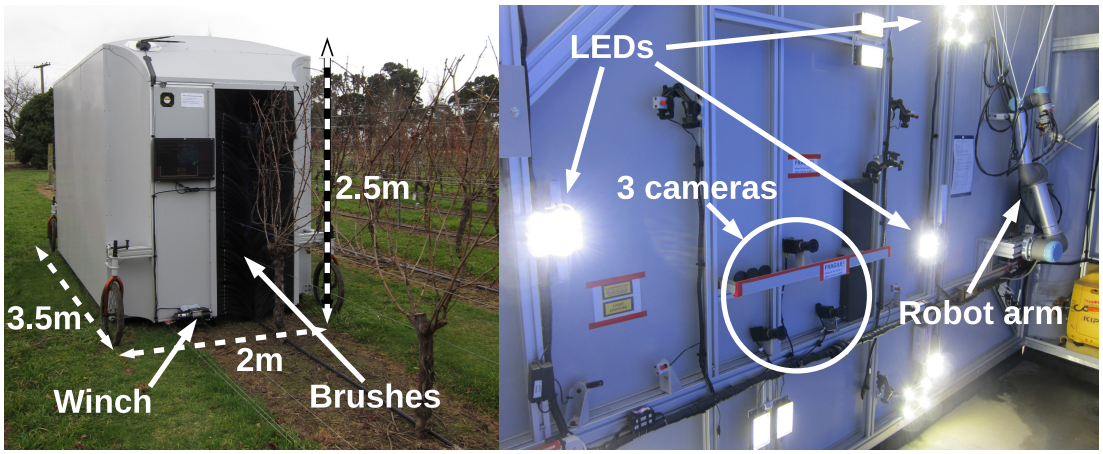
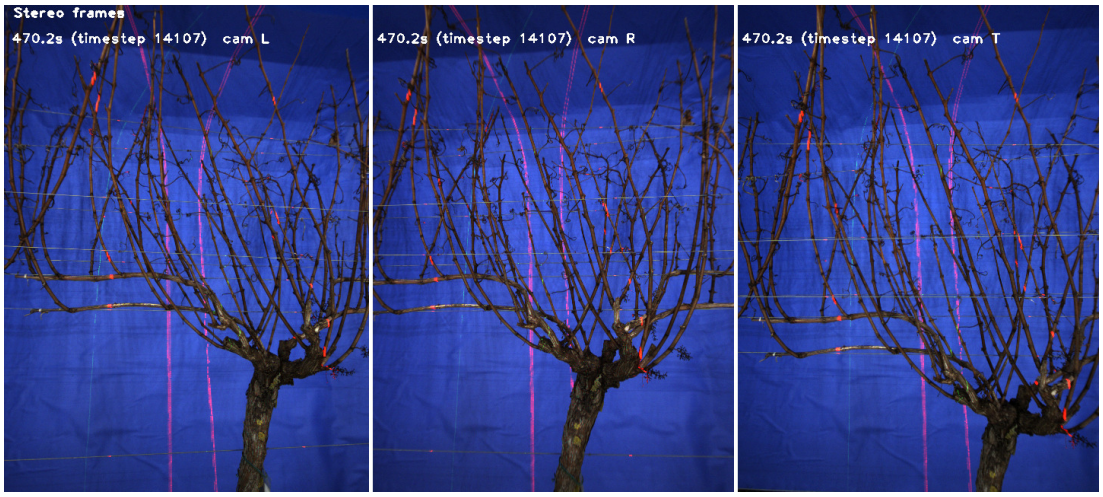
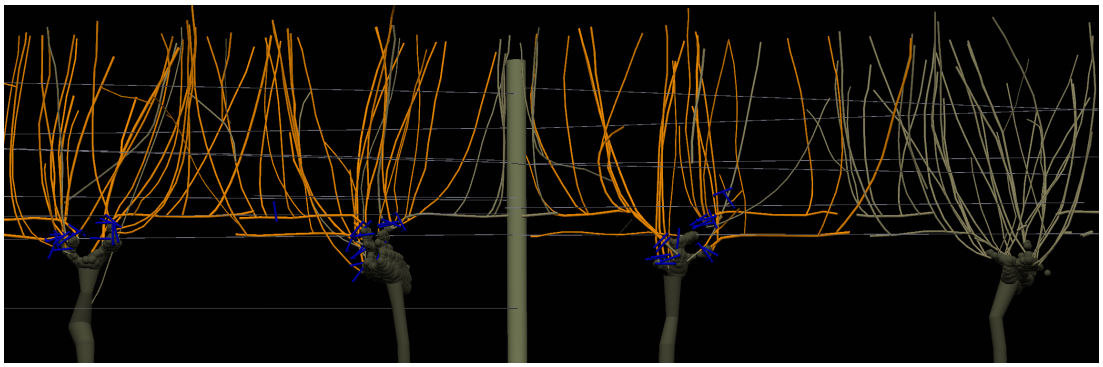


Figure 1.1: Vine pruning robot [Botterill et al 2016].



(a) Grape vines to be reconstructed.



(b) Reconstructed grape vines. Optimal cuts are marked in blue and vines to be removed are shown in orange.

Figure 1.2: 3D reconstruction of grape vines [Botterill et al 2016].

Figure 1.3: Robot arm making a cut [Botterill et al 2016].

1.1 ROBOT ARM PATH PLANNING

Path planners use sensor data to compute a path for the robot to achieve its task without colliding with the environment or itself. This path is then parameterised by the relative time that the robot will reach each waypoint, forming a trajectory. The trajectory is then executed by the robot, and the environment may be altered if objects are moved (Fig. 1.4).

To speed up computation time, path planners run on robots without differential constraints, e.g. many robot arms, ignore the robot's velocity and accelerations. These velocities and accelerations are computed to minimise the execution time of the path after the planner has terminated with a valid path (Fig. 1.5). Quick-to-execute paths can still be found by the planner because execution time is often strongly correlated with the path's Euclidean length (see the figures in Chapt. 6), even though other factors such as the accelerations of the robot's joints influence the execution time of the path.

Robot arms are typically required to perform tasks in their *workspace*, e.g. reach the fruit at a certain Cartesian position. The workspace of a robot arm is typically two or three dimensional and consists of all the points reachable by the robot. Alternatively, the robot's position can be represented by its joint angles, or *configuration*. A robot's *configuration space* describes all sets of joint angles that can be obtained by the robot.

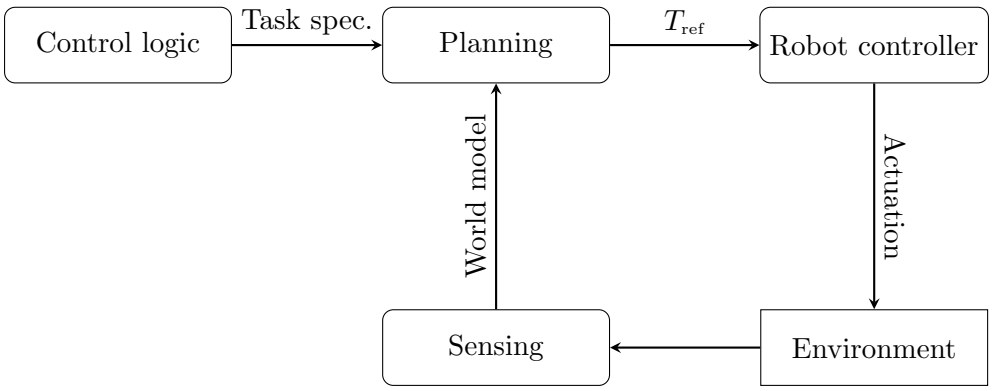


Figure 1.4: How the path planning module typically fits in with the rest of a robotic system. Boxes with rounded corners are software systems within the robot. Boxes with square corners are external to the robot. T_{ref} is a reference trajectory.

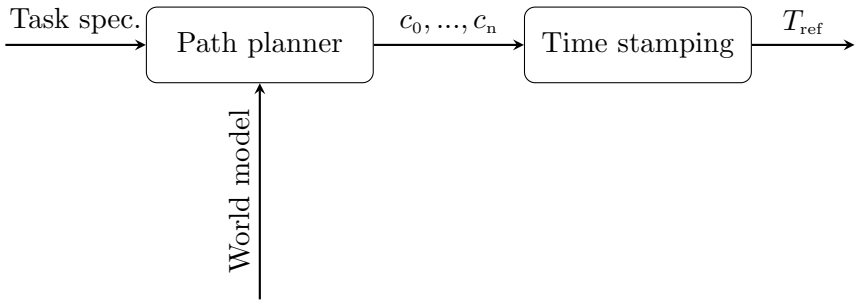


Figure 1.5: Planning node from Fig. 1.4. T_{ref} is a reference trajectory and $c_0, ..., c_n$ denotes a sequence of robot configurations.

Forward and Inverse kinematics are used to map the robot's pose in its workspace to its configuration. Forward kinematics maps a configuration to link poses in the workspace. This can often be solved analytically for serial manipulators. Inverse kinematics maps the pose of one or more links to configuration space. Solutions can be found using numerical optimisation algorithms such as Levenberg–Marquardt [Marquardt and Donald. W. 1963]. Analytical solutions exist for some robot arms e.g. the UR5 [Hawkins 2013]. One end-effector pose for the robot can sometimes be achieved by the robot in more than one configuration (Fig. 7.1).

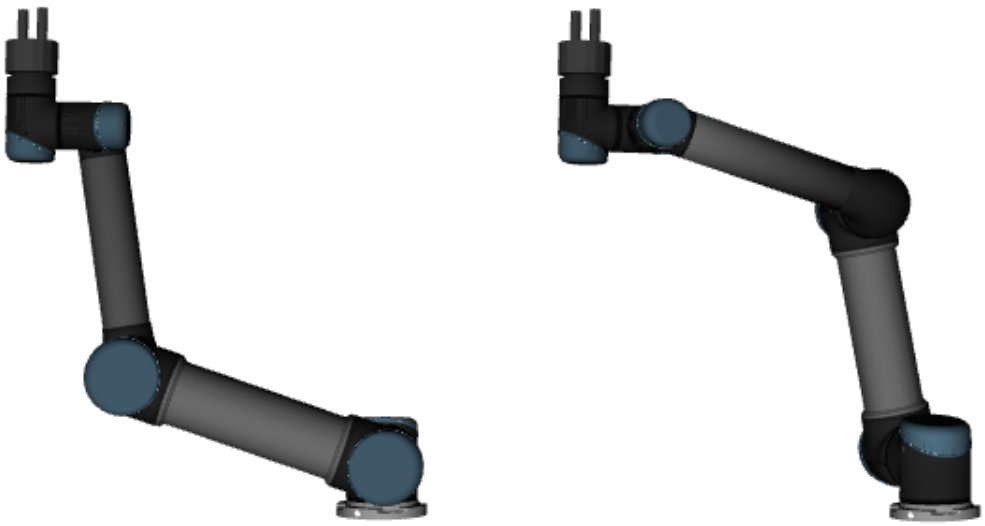


Figure 1.6: Two poses for the UR5 robot arm that put the end-effector at the same position.

1.2 MOTIVATION

Enabling robots to perform their tasks quickly is an important problem for many different robot systems. Bac et al [2014] use cycle time as a performance indicator in their review of 50 recent robotic systems for harvesting fruit, and cite it as an important economic factor for agricultural robots.

The primary motivation of this thesis is to develop new approaches for path planning that allow the vine pruning robot to quickly prune vines. To achieve this, my approaches should quickly compute high quality (e.g. short) paths for the robot arm. The new approaches developed in this thesis can be used on different robots. Some of these approaches are tested on a secondary robot task as well as being tested on the vine pruning robot.

1.3 THESIS ORGANISATION

Chapter 2 contains a review of the path planning literature relevant to agricultural robots that use a robot arm. This covers approaches that have been used to speed up and/or improve the quality of paths found for robot arms. It also covers approaches for dealing with 'real world' effects such as sensor uncertainty.

Chapter 3 describes the experiment setup. This details about the input data, how the robot arm was controlled, how cut positions were selected and how cuts were performed. A secondary experiment, a robot for reaching into cubicles, is also described. This secondary robot was used for testing the proposed approaches in Chapters 6 and 7 to verify that they were applicable to robotic tasks other than grape vine pruning.

Chapter 4 presents a collision detector that uses the structure of grape vines to perform fast collision detection. This fast collision detector reduced mean path planning times from 1.83 seconds to 0.076 seconds (a 28 times improvement) using the RRT-Connect planner when compared to using the Flexible Collision Library (FCL) [Pan et al 2012].

In Chapter 5 the performance of 17 commonly used sampling based path planners is compared. The most successful planners for this task are identified, and one of them is selected to be further improved in subsequent chapters.

Chapter 6 presents a path planning algorithm that integrates an asymptotically optimal path planner with a local optimiser to speed up the search for high quality paths. By integrating a short-cutting local optimiser the planner was able to find paths that had a 31% shorter execution time when converted to trajectories after three seconds of computation time.

Many robot arms can achieve their tasks using more than one goal configuration, in this thesis these arms are called redundant. Chapter 7 shows how these extra goal configurations can be used by an asymptotically optimal path planner to quickly find high quality paths. By using these extra goal configurations, paths that had a 58% lower execution after being converted to trajectories could be found.

Chapter 8 contains conclusions and discusses future work. Additional material, e.g. specification sheets, are contained in the appendices.

Chapter 2

PATH PLANNING LITERATURE REVIEW

Path planners often operate in the robot's configuration space [Lozano perez 1983] to find collision free paths. A configuration represents the position of each of the robot's joints, this is six dimensional for the six degree of freedom UR5 robot arm used for vine pruning. The configuration space, C, can be split into C_{free} and C_{obs}. C_{free} is the set of all configurations where the robot is not in collision with the environment or itself. C_{obs} is the set of configurations where the robot is in collision with itself or the environment. Computing an explicit representation of configuration space for many robot arms is prohibitively expensive.

Sampling-based planners [Kavraki et al 1996, LaValle 1998, Sucan and Kavraki 2010, Sucan et al 2012] are by far the most widely used methods used for online planning for robot arms and other high degree of freedom robots because they do not require an explicit representation of the robot's configuration space. They use a collision detector to classify sampled configurations as either in C_{free} or C_{obs}. Artificial Potential Fields [Khatib 1986, Newman and Hogan 1987, Hwang et al 1992, Barraquand et al 1992] and geometry-based methods [Lozano prez and Wesley 1979, Schwartz and Sharir 1983] have also been developed, but are limited to simple environments [Koren and Borenstein 1991] because they get stuck in local minima or require an explicit representation of the robot's configuration space.

Path planners can be categorised as feasible planners [Hsu et al 1999, Kavraki et al 1996, LaValle 1998, LaValle 2001], or optimizing planners [Gammell et al 2015, Janson et al 2015, Karaman and Frazzoli 2011, Salzman and Halperin 2016]. Feasible

planners attempt to quickly find a solution and terminate as soon as one is found. Feasible planners can return poor, e.g. long, solutions because they do not optimize solutions. Optimizing planners attempt to find high-quality, e.g. short, solutions within a set computation time or number of iterations.

Many sampling based path planners are *probabilistically complete* [LaValle 1998, Kuffner and Lavalle 2000, Sucan and Kavraki 2009a]. As the number of samples drawn by the planner approaches infinity the probability that a solution is found, given a robustly feasible solution [Karaman and Frazzoli 2011] exists, approaches 1.

Some sampling-based path planners are also *asymptotically optimal*. The cost of the best path found by an asymptotically optimal planner will approach the cost of the optimal path, given a robustly optimal path [Karaman and Frazzoli 2011] exists, as the number of iterations approaches infinity.

Sampling-based path planners often use an acyclic graph data-structure referred to as a tree. Vertices in the graph represent sampled configurations and edges represent local paths between these configurations. These local paths can be straight lines in configuration space, e.g. when planning for a robot arm, or curved arcs when planning for car robots. The planner returns a path, which is a sequence of vertices and edges connecting the start and goal configurations. Graph vertices are fast to collision check because they represent a single point in configuration space, while edges can be expensive to check because they represent a path.

Some path planners are designed to quickly compute one collision free path (single-query planners), while others can reuse computation from previous queries (multiple-query planners). Single-query planners often maintain a tree data-structure where each vertex is only connected to one other vertex. Limiting the number of connections means that single-query planners can perform fewer time consuming collision checks than multiple-query planners. Multiple-query planners often maintain a graph structure where each vertex may be connected to a set number of other vertices, e.g. k-PRM [Kavraki et al 1996] or k-PRM* [Karaman and Frazzoli 2011], or all visible vertices within a radius e.g. r-PRM [Kavraki et al 1996], r-PRM* [Karaman and Fraz-

zoli 2011] or Sparse Roadmap Spanners (SPARS) [Dobson and Bekris 2014].

Multiple-query planners, e.g. Probabilistic Roadmap (PRM) [Kavraki et al 1996], often construct a graph with many edges between vertices as shown in Fig. 2.1, this is often referred to as a *roadmap*. It should be noted that straight lines in configuration space do not represent straight line motions in the workspace, for a robot arm these often result in arcing motions. The roadmap is expanded during planning as shown in Fig. 2.2. The routine *GrowRoadmap* expands the roadmap, and *ShortestPath* returns the shortest path through the roadmap between two vertices e.g. by using A* graph search [Hart and Nils 1968]. Multiple query planners are effective when they are used for more than one query in an environment because they can re-use the roadmap. Roadmaps can be slow to construct because they often have a large number of edges that need to be collision checked. This means multiple-query planners can be slow if they are only required for few planning queries.

The PRM is the most well known multiple-query sampling based path planner. To solve a planning query the PRM grows its roadmap until the start and goal configuration are within one connected component as shown in Fig. 2.2. The PRM roadmap is typically grown by batch-sampling and then connecting these new samples into the roadmap as shown in Fig. 2.3. The *BatchSampleVertices* method returns a number of collision-free vertices, where the exact number might be a parameter set by the user. The original implementation used a batch size of one. The *ConnectNewSamples* routine forms collision-free edges between the newly sampled vertices and the existing roadmap. Some variations of PRM, e.g. k-PRM[1], attempt to connect each new vertex to the nearest k vertices. Other variations, e.g. r-PRM, attempt to connect new vertices to other vertices within a radius. The original PRM implementation only attempted to connect vertices if they were in different connected components. When the start and goal vertices are in the same connected component a graph search algorithm, e.g. A*, is used to recover the path between the start and goal vertices.

Single query planners are effective at quickly finding collision free paths. These

[1]It should be noted that k-PRM is not always probabilistically complete [Karaman and Frazzoli 2011]

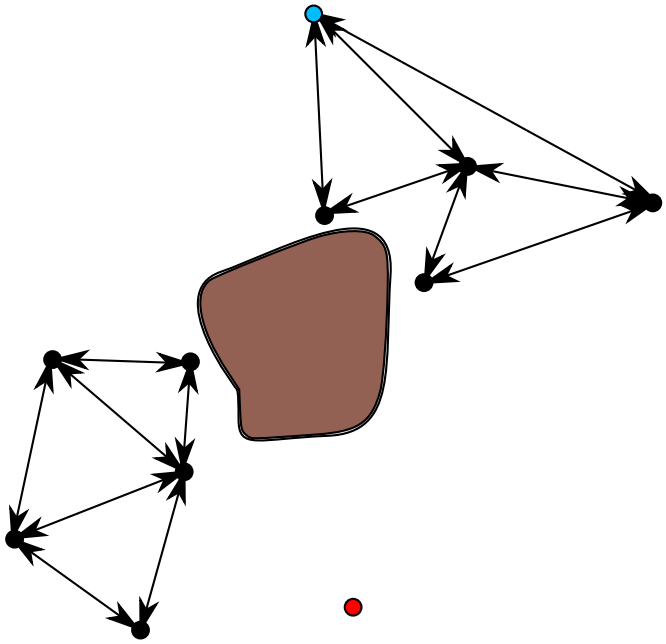


Figure 2.1: Roadmap with two connected components and a brown obstacle. Filled circles show vertices, arrows show edge directions. The query start vertex is shown in blue and goal vertex is shown in red.

```
1: function QUERY(G = (V, E), q_start, q_goal)
2:     do
3:         G ← GrowRoadmap(G)
4:     while q_start and q_goal are not in the same connected component
5:     return ShortestPath(G, q_start, q_goal)
6: end function
```

Figure 2.2: An example of a multiple query planner searching for a path from q_{start} to q_{goal} through its roadmap G.

```
1: function GROWROADMAP(G = (V, E))
2:     V_sampled ← BatchSampleVertices()
3:     G ← ConnectNewSamples(G, V_sampled)
4:     return G
5: end function
```

Figure 2.3: GrowRoadmap procedure for the PRM path planner.

planners build a directed tree as shown in Fig. 2.4. The vertices of this tree represent points in the robot's configuration space and edges in this tree represent paths between these vertices. This tree has few edges because each vertex is only connected to one other vertex. Having few edges makes trees relatively quick to construct because fewer expensive edge collision checks need to be performed. However, having few edges means that trees are not usually useful for planning queries other than the one they were constructed for. Fig. 2.5 shows a procedure for solving a planning query with a single query planner. The function *GrowTree* expands the planner's tree. *Trace* constructs the path between the start and goal vertex by recursively following each parent's vertex starting with the goal.

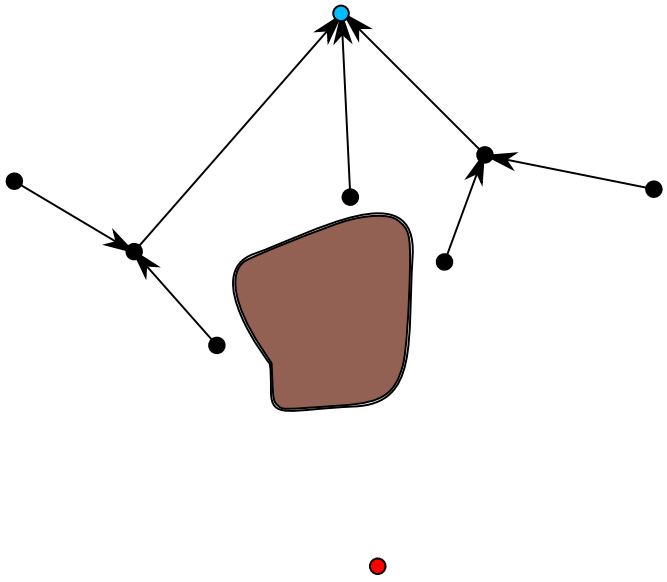


Figure 2.4: Tree of motions rooted at the blue circle with a brown obstacle. Filled circles show vertices, arrows show edge directions. The goal vertex is shown in red.

The Rapidly Exploring Random Tree (RRT) [LaValle 1998] is the most well known single-query planner. It incrementally grows a tree to find a collision free path from the start vertex to the goal vertex. The maximum edge length is controlled by a parameter often referred to as *range*. Fig. 2.6 outlines the expansion process. A new vertex is sampled from the robot's configuration space. To speed up planning this new vertex will be equal to the goal vertex a set fraction of the time. If this vertex is further than *range* from its nearest neighbour in the tree then interpolation is performed and the vertex is moved to be a distance of *range* from the tree. If the motion from the tree

```
1: function QUERY(q_start, q_goal)
2:     Initialise G to be a graph with a single vertex v_start at q_start
3:     v_goal ← q_goal
4:     do
5:         G ← GrowTree(G, v_goal)
6:     while v_goal ∉ G
7:     return Trace(G, v_goal)
8: end function
```

Figure 2.5: An example of a single query planner finding a path from q_{start} to q_{goal}. GrowTree may take additional parameters.

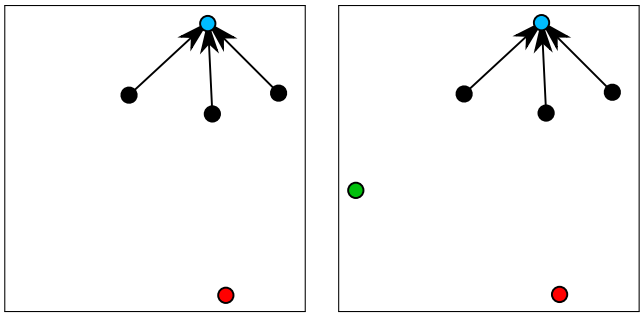
to the new vertex is collision free then an edge to the new vertex is formed and it is added to the tree.

Planning from experience approaches combine the fast plan times of single-query planners with the ability to reuse computation. A single-query planner is used to quickly compute collision-free paths for planning queries, and these paths are stored in a graph [Phillips et al 2012, Coleman et al 2015] or a path database [Berenson et al 2012]. Eventually some planning queries can quickly be satisfied by using data from the graph or database, without needing to invoke the single-query planner.

2.1 ASYMPTOTICALLY OPTIMAL MOTION PLANNING

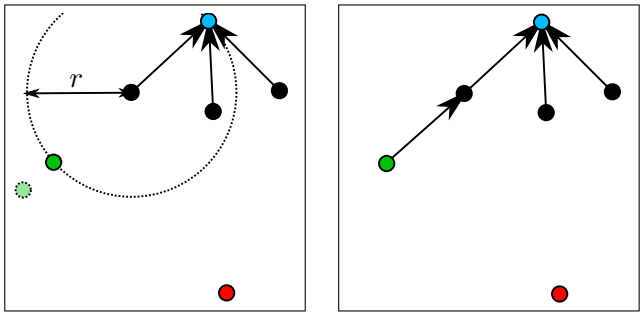
Asymptotically optimal planners eventually converge to the optimal solution [Karaman and Frazzoli 2011]. They are similar to probabilistically complete feasible path planners, but the edges of all vertices must remain optimal within a local *neighbourhood*, see Appendix 8.2. A number of feasible path planners have been adapted to be asymptotically optimal (Tab. 2.1).

On the insertion of new vertices, asymptotically optimal multi-query planners, e.g. PRM* [Karaman and Frazzoli 2011], must form edges between the new vertex and all other vertices within its neighbourhood. Asymptotically optimal single query planners, e.g. RRT* [Karaman and Frazzoli 2011], connect a new vertex to the neigh-



(a) Tree of motions.

(b) New vertex is sampled (green).



(c) Vertex is saturated.

(d) Vertex is added to tree.

Figure 2.6: Expansion process for RRT with range r. The start vertex is shown in blue, the goal vertex is shown in red. The green vertex is sampled, saturated, and added to the tree.

Table 2.1: Asymptotically optimal variations of feasible path planners.

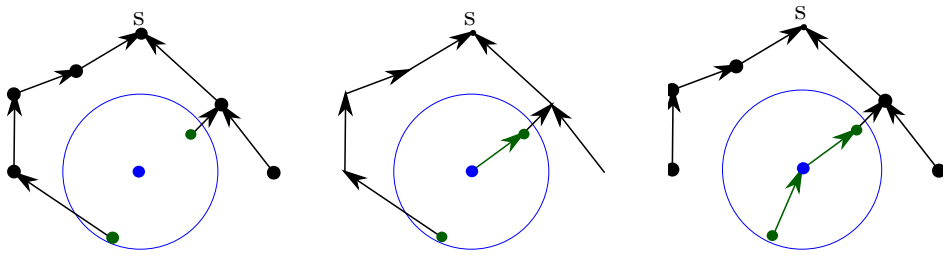
Feasible planner	**Optimal variation**
RRT	RRT* [Karaman and Frazzoli 2011]
RRTConnect	RRTConnect* [Akgun and Stilman 2011, Klemm et al 2015, Jordan and Perez 2013]
PRM	PRM* [Karaman and Frazzoli 2011]
TRRT	TRRT* [Devaurs et al 2016]

bouring vertex that minimises the cost to arrive at the new vertex. The edges from the remaining vertices in the neighbourhood must then be updated if the new vertex provides a better path to arrive as shown in Fig. 2.7. When a new vertex, v_{new} is sampled its neighbourhood is calculated as either all vertices within a radius or the nearest k vertices (Fig. 2.7a). An edge is then formed between v_{new} and the vertex in the planner's tree that minimises the cost to arrive at v_{new} (Fig. 2.7b). Edges to other vertices within the neighbourhood may also be updated (Fig. 2.7c). Updating or forming edges is expensive because it requires collision checking.

Some optimizing planners are asymptotically optimal [Karaman and Frazzoli 2011] and will eventually converge to the optimal solution [Gammell et al 2015, Janson et al 2015, Karaman and Frazzoli 2011]. Other optimizing planners are asymptotically near-optimal and will eventually converge to a near-optimal solution [Arslan 2013, Dobson and Bekris 2014, Otte and Frazzoli 2015, Salzman and Halperin 2016].

2.1.1 Batch sampling

Optimal sampling-based path planners can be split into those that perform incremental search, and those that perform batch sampling. RRT* [Karaman and Frazzoli 2011] and other incremental sampling based path planners attempt to add one vertex to their tree of motions at each iteration. This makes them well suited to finding fine motion plans in large configuration spaces.



(a) New vertex is sampled and its neighbourhood is calculated.

(b) New vertex is added to the planner's tree.

(c) Neighbouring vertices are updated.

Figure 2.7: Locally optimal insertion of a new vertex (blue) into an incremental planner's tree of motions. The blue circle shows the new vertex's neighbourhood and green arrows show updated edges. Euclidean path length is being minimised.

Batch sampling based single query motion planners, e.g. Fast Marching Trees (FMT*) [Janson et al 2015] and Batch Informed Trees (BIT*) [Gammell et al 2015], compute a batch of collision-free samples and then attempt to find high quality collision free paths through these samples. Unlike incremental planners, batch planners do not modify samples to bring them closer to the planners graph (or tree). The tree is then constructed in a lazy fashion, where vertices are added in order from most promising to least promising. Vertices that cannot be used to find a lower cost, e.g. shorter, path through the graph than the current best path are not added, saving computation time. These planners can have faster convergence times than incremental planners in some problems [Gammell et al 2015, Janson et al 2015, Starek et al 2015] because they use this information about how promising a state is to guide the search.

Batch sampling planners are not suited to finding fine motion plans in large configuration spaces. This is because the granularity of the plan that they can find is governed by how densely the configuration space is sampled. Dense sampling of large configuration spaces can require a large number of samples to be generated and collision checked. This large number of samples can also take a long time to process. BIT* partially remedies this by drawing multiple smaller batches of samples from subsets of configuration space that decrease in size. This biasing is well suited to problems where an initial solution can be quickly found and the optimal solution is a close to the straight line from the start to the goal. Having a smaller batch size limits the planner's ability to only add promising states into it's search graph. In this thesis incremental sampling-based path planners are used because the vine pruning robot requires fine motion plans to get the cutting tool close to cut points that are very close to obstacles (other parts of the plant). A plan resolution of approximately 1cm in the workspace may be required for areas near cuts.

2.1.2 Focussed search heuristic

The convergence speed of optimizing planners can be limited by their ability to draw useful samples [Gammell et al 2014]. As the planner converges to the optimal solution

it spends more time processing samples that cannot possibly be used to improve on the best solution [Gammell et al 2014]. This can be remedied by focussing the planner's search to useful regions of configuration space [Gammell et al 2014] and rejecting new samples that cannot be used to improve on the planner's best solution [Akgun and Stilman 2011] without sacrificing the optimality properties of the planner.

The region of the robot's configuration space where samples can be used to improve the best path can be defined as:

$$X_f = \{x \in X | f(x) < c_{\text{best}}\}, \tag{2.1}$$

where x is a configuration in the robots configuration space X, this is a six dimensional point for the UR5. c_{best} is the cost of the best path found so far by the planner. $f(x)$ is the optimal cost of a collision free path that goes from the start configuration to the goal through x. In practice, $f(x)$ can be approximated by the admissible cost of x. The admissible cost never overestimates $f(x)$ and is specific to the optimization objective being used[2]. An informed sampler [Gammell et al 2014] can be used to draw samples directly from X_f once an initial solution has been found.

The focussed search procedure is outlined in Fig. 2.8. The informed sampler samples vertices from within X_{f} which is shown as an ellipse. Even though the newly sampled green vertex is within X_{f} it cannot possibly be used to find a shorter path than the dark blue one that has already been found. The green vertex is rejected according to the heuristic in Akgun and Stilman [2011].

2.2 BIDIRECTIONAL SEARCH

Some path planners use a bidirectional search to find collision free paths faster [Kuffner and Lavalle 2000, Sucan and Kavraki 2010, Hsu et al 1999, Sánchez and Latombe 2003a], and/or speed up convergence [Akgun and Stilman 2011, Jordan and Perez 2013, Klemm et al 2015, Starek et al 2015]. Two trees are grown toward each other, one from the start

[2]The straight line path length can be used as an admissible heuristic when minimising path length

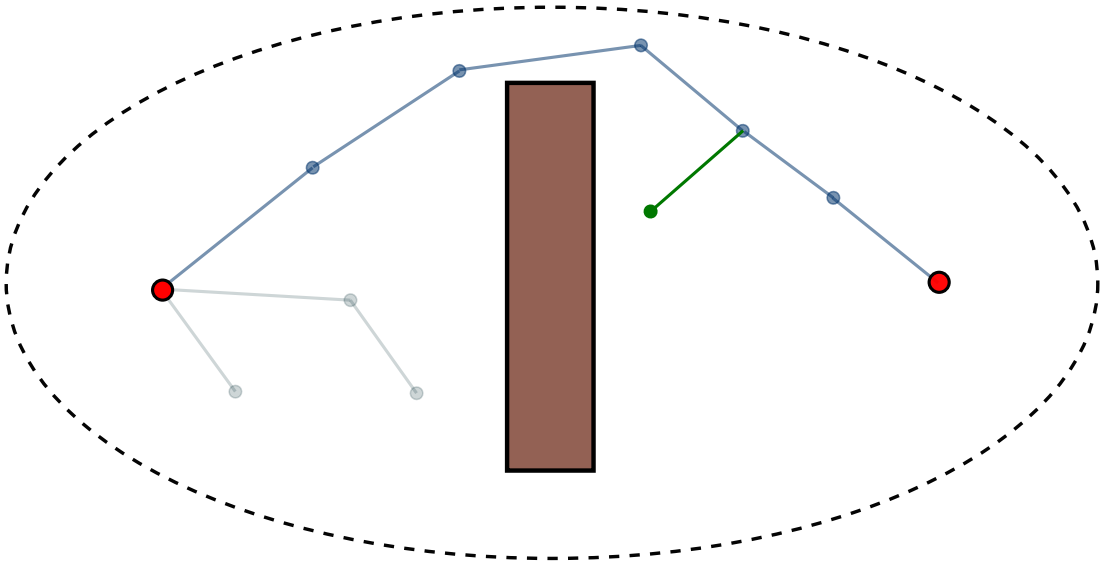
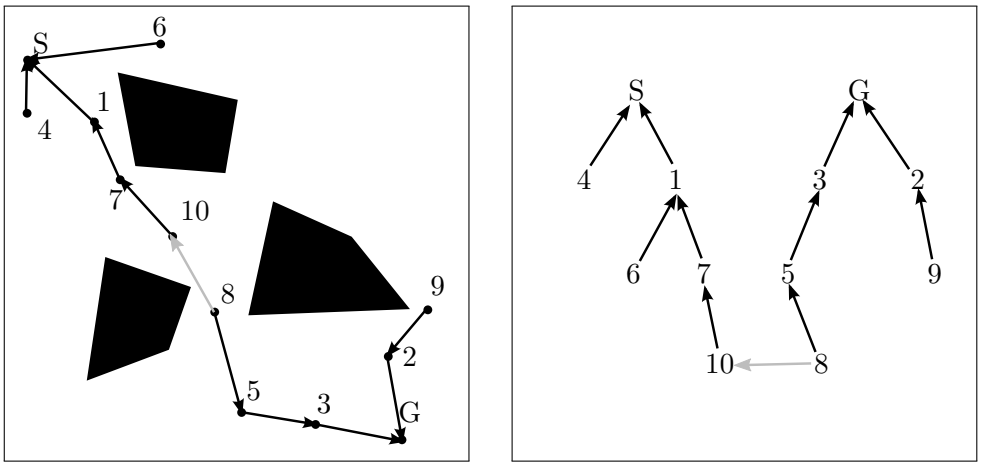


Figure 2.8: Focussed sampling after an initial solution (dark blue) has been found. The brown rectangle represents an obstacle. The dashed ellipse shows the region X_f.

state and one from the goal state. Paths are found when the two trees are connected (Fig. 2.9). Bidirectional search works well when the goal configuration is difficult to reach, e.g. is in a cluttered part of configuration space. This is because the goal tree is a bigger target for the start tree than a single configuration.

Some bidirectional planners attempt to join the trees when they are close, e.g. the bidirectional Fast Marching Trees (BFMT*) [Starek et al 2015], or whenever a new vertex is added to one of the trees, e.g. RRTConnect [Kuffner and Lavalle 2000]. Both strategies scale favourably with configuration space dimension [Starek et al 2015, Luo and Hauser 2014]. The greedy connection strategy of RRTConnect is particularly well suited to configuration spaces where there are large regions of free space between the trees. RRTConnect is particularly useful for vine pruning where fine motions are required around the start and goal, e.g. to avoid canes around the cuts, but where there are large free regions of configuration space between the trees, e.g. when the robot arm pulls back from the plant. This makes bidirectional planners a good fit for robot arm path planning, because robot arms often have large configuration spaces and require fine motion planning around the start and goal.



(a) Path plan from a bidirectional search. Connecting motion is shown in light gray.

(b) Start (S) and goal (G) trees of motions with the connecting motion in light gray.

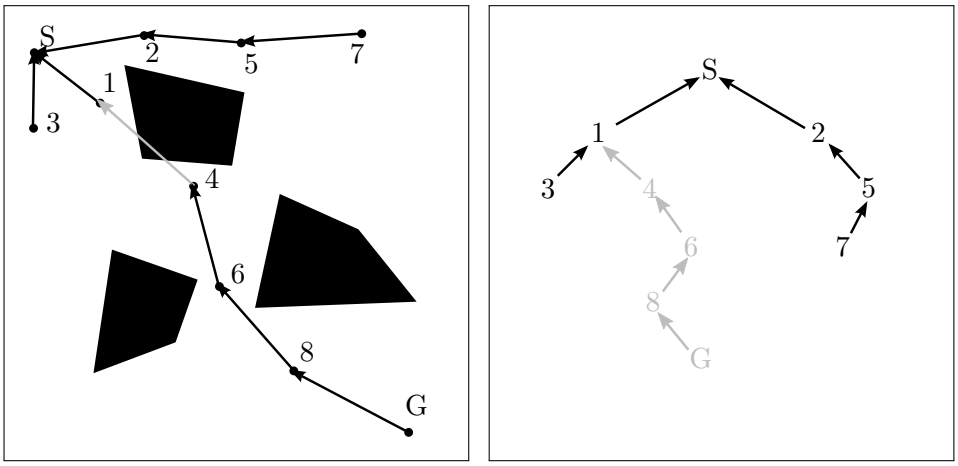
Figure 2.9: Path plan and corresponding tree of motions for a bidirectional search between the start (S) and goal (G) vertices.

2.3 LAZY COLLISION CHECKING

A large amount of path planning time is spent collision checking edges that will not be part of the final solution. Some single query and multiple query planners attempt to speed up planning by delaying the collision checking of edges until a path is found between the start and the goal (but vertices are still checked) [Bohlin and Kavraki 2000, Bohlin and Kavraki 2001, Gasparri et al 2009, Sánchez and Latombe 2003a], these planners are referred to as 'lazy'. Once a path is found, the edges are collision checked. If they are all collision free the planner can terminate with a solution, otherwise edges that are in collision are pruned from the planer's graph. Lazy single query planners will additionally prune all descendants of invalid edges to prevent adding disconnected components (Fig. 2.10).

A number of lazy planners have been developed for feasible, e.g. Lazy-PRM [Bohlin and Kavraki 2000] and SBL[3] [Sánchez and Latombe 2003a], and asymptotically (near) optimal planning, e.g. Lazy-PRM* [Hauser 2015] and Lazy-LBTRRT [Salzman and Halperin 2016]. Choudhury et al [2016a] extend the Lazy-PRM* to evaluate paths that are less likely to be in collision first to speed up the search for initial solutions.

[3]Single-query bidirectional path planner with lazy collision checking.



(a) Lazy path plan with an invalid path between milestones one and four.

(b) Tree of motions corresponding to lazy path plan. Motions and milestones in light gray are to be removed due to the invalid motion between milestones one and four.

Figure 2.10: Lazy path plan between the start (S) and goal (G) vertices with an invalid motion.

Planners with lazy collision evaluation perform well when there is a small chance of edges being in collision. This mostly depends on the configuration space being planned in and the length of edges in the planner's graph, which can often be controlled with a *range* parameter. If too many edges are in collision the planner will often have to discard many edges and restart planning with a smaller graph. If few edges are in collision then a lazy planner is more likely to find a collision-free solution earlier than a non-lazy planner because it performs less collision detection.

2.4 SAMPLE BIASING

Sample biasing is a common approach for improving computation time of feasible path planners and the convergence time of optimizing planners. Goal biasing is the most common form of sample biasing. Many incremental single-query planners will sample a goal state with a constant probability to encourage growth toward the goal [LaValle 1998, Otte and Frazzoli 2015, Hsu et al 1999, LaValle 2001, Ladd and Kavraki 2005].

A number of non-uniform strategies have been proposed for use with batch sampling planners, e.g. PRM. Gaussian [Boor et al 1999] and obstacle based sampling [Amato

et al 1998] are used to generate samples near obstacles. The bridge test for narrow passages [Hsu et al 2003] and medial axis sampling to maximise obstacle clearance [Wilmarth et al 1999]. There is some contention about the effectiveness of these approaches, with Geraerts [2006] finding that they only work well for specific problems and can significantly degrade the planner's performance in other problems.

Thomas et al [2007] also identify that certain sampling schemes are only effective in certain problems. They propose a framework for combining multiple sampling schemes, where each sampler biases the sample returned by the previous sampler. Although this approach allows the strengths of different samplers to be combined, the planner's performance is now dependant on the *combination* of samplers that is used. Picking a good combination of samplers for a particular problem may be difficult in practice.

Zhong and Liu [2016] propose a hybrid sampling scheme that combines a uniform, a Gaussian, and a bridge test sampler. The configuration space is split into regions by a classifier. Samples for each region are drawn independently from the three samplers in different proportions as dictated by the classifier. Ideally the individual samplers get used in local regions that pose problems which they were designed for e.g. the bridge sampler gets used in regions with narrow passages. Regions where it is difficult to sample collision-free configurations are additionally sampled more heavily than those where collision-free configurations can be sampled.

Both Thomas et al [2007] and Geraerts [2006] agree that connecting samples, rather than generating samples in difficult to sample regions of configuration space, is a more important problem in motion planning for batch sampling planners. Incremental planners, e.g. RRT, are better at joining samples because they relocate new samples to be a maximum of *range* from the planner's graph. However, this also means that sample biasing is difficult to achieve with incremental planners because the samples may be moved by the planner.

Path biased sampling is sometimes applied to optimising planners in order to make them more exploitive, and can be applied to both incremental and batch sampling planners. Alterovitz et al [2011] bias sampling toward newly discovered path in their

variation of RRT*. Nasir et al [2013] implement a more exploitive variation of the approach by Alterovitz et al [2011] with their RRT*-Smart planner. RRT*-Smart biases sampling toward *only* the best path found so far. RRT*-Smart is particularly effective compared to RRT* when there are few, e.g. 1-3, homotopy classes in the problem as shown in the results in Nasir et al [2013]. Alterovitz et al [2011] perform few experiments with their biased RRT* planner, but it should outperform RRT*-Smart on problems with a large number of homotopy classes because it is less greedy.

2.5 PLANNING WITH UNCERTAINTY

In real robots, the environment generated through sensor data is not the same as the real world model. This results in path plans being computed that appear collision-free, but in fact result in collisions when used on a real robot. To overcome this, path planners can incorporate information about the certainty of obstacle locations [Chakravorty and Kumar 2011, Bopardikar et al 2016, Valencia et al 2010]. Errors can sometimes be modelled by Gaussians [Missiuro and Roy 2006] when this information is not known. Sometimes Gaussians are also a poor model, e.g. for errors resulting in incorrect correspondences in the 3D reconstruction.

When errors are difficult to model, one can use an optimizing planner such as RRT* to maximise obstacle clearance. This could be computationally expensive due to the slow convergence of RRT* and the extra collision detection time required to compute distances to nearest obstacles.

Another approach is to introduce a safety margin by increasing the obstacle size [Fahimi 2009]. This enforces a minimum clearance in computed plans from the model of the environment. It works well because introducing a safety margin will make plans more robust for almost any error model. Problems arise when goals are close to obstacles, e.g. for vine pruning where cut-points are on obstacles, and it becomes difficult or impossible to find a collision free configuration that satisfies the goal constraint.

2.6 USING OBSTACLE PROXIMITY TO REDUCE COLLISION CHECKING

Some approaches to path planning store obstacle proximity information and use it to speed up, or avoid, future collision checks. This information is often stored as a volume in the robot's configuration space or workspace that is known to be collision free. Approaches that store configuration space approaches, e.g. configuration space safety certificates [Bialkowski et al 2016], can only be used when the configuration space distance between a configuration and the nearest obstacle can be computed, e.g. when an explicit representation of the obstacles in configuration space is known. This is very difficult when an explicit representation of obstacles is not known e.g. for robot arm path planning.

In many cases, e.g. planning for a robot arm, this is very difficult.

Collision free volumes in the workspace can be computed by enlarging the robot model [Vahrenkamp and Asfour 2007, Bialkowski et al 2016] or fitting other volumes. A common approach is to fit bubbles around robot poses in the workspace. Early approaches compute very conservative bubbles that only extend to the nearest obstacle [Quinlan and Khatib 1993, Bertram et al 2006, Lacevic and Rocco 2010]. These bubbles could be very small in cluttered environments when the distance to the nearest obstacle is small in at least one direction. Later approaches find less conservative volumes by performing dilations [Bialkowski et al 2016] or finding free workspace 'slices' [Ademovic and Lacevic 2016]. Finding large collision free volumes is advantageous because it may mean that more collision checks can be bypassed. Workspace collision-free volumes can then be used for guiding the planner's search as well as bypassing some collision checking [Lacevic and Rocco 2013, Ademovic and Lacevic 2014, Lacevic et al 2016].

2.7 PATH OPTIMISATION

Local optimization algorithms can be used with path planners to quickly improve path quality, e.g. length. These algorithms often rely on a path planner to provide an initial

solution. This initial solution influences the quality of the optimised path because these algorithms only optimize locally. Short-cutting [Berchtold and Glavina 1994, Geraerts et al 2007, Hauser and Ng thow hing 2010] for reducing path length and sequential convex optimization approaches [Kalakrishnan et al 2011, Schulman et al 2014, Zucker et al 2013] have been shown to work well on robot arms.

A common approach to finding short paths is to find an initial collision-free solution with a feasible planner, e.g. RRTConnect [Kuffner and Lavalle 2000], and to optimise this path with a local optimiser e.g. short-cutting. Another approach is to perform multiple restarts of the feasible planner, optimise each solution and return the best solution. This has been shown to work well in empirical experiments when compared to asymptotically optimal planners [Luo and Hauser 2014].

After short-cutting, paths will tend to come very close to obstacles. This is fine in simulation, but poses problems when used with a real robot that has error in the environment model. On the vine pruning robot there are errors in the 3D reconstruction which means paths that appear valid to the planner result in collisions between the robot arm and obstacles. One method to mitigate this is by adding a safety margin to the obstacles, modelling them as larger than they are [Fahimi 2009]. This approach is not suitable for vine pruning out-of-the box because the targets for the cutting tool are obstacles for the rest of the robot. This could mean that with added thickness to obstacles the targets cannot be reached without collision.

Path smoothing is performed after planning and short-cutting. This prevents jerky robot motions. A common implementation is to use interpolation and short-cutting on subdivided paths [Sucan et al 2012].

2.8 PATH PLANNING WITH REDUNDANT MANIPULATORS

Robot arms are often required to perform tasks in their workspace, e.g. move the end-effector into a specific pose. These tasks can often be accomplished by the arm by using many different configurations.

```
1: function RRT(start, goal, range, goal_bias)
2:     G ← start
3:     do
4:         rand ← A random number in [0,1)
5:         if rand < goal_bias then
6:             v_sampled ← A configuration sampled from goal
7:         else
8:             v_sampled ← A random configuration
9:         end if
10:        v_near ← The closest vertex in G to v_sampled
11:        // Set v_sampled to be at most range from v_near.
12:        v_sampled ← Interpolate(v_near, v_sampled, range)
13:        if The path from v_near to v_sampled is collision-free then
14:            e ← The edge from v_near to v_sampled
15:            G ← G ∪ e, v_sampled
16:        end if
17:    while G does not contain a vertex that satisfies goal
18:    v_goal ← The vertex in G that satisfies goal.
19:    return Trace(G, v_goal)
20: end function
```

Figure 2.11: RRT with an abstracted goal representation.

Some approaches for path planning with redundant robot arms rely on using a goal representation other than a single configuration. The representation of a goal for a sampling based path planner, such as RRT, can be abstracted as shown in Fig. 2.11. The *goal* variable may represent a single configuration, multiple configurations or anything else that can have configurations sampled from it.

In many cases the goal is represented by one configuration [Lee et al 2014, Coleman et al 2015, Stilman et al 2007, Hirano et al 2005] that satisfies the robot's task. This means that the path planner is constrained to finding a collision free path that ends at one particular configuration, when there may be many, possibly better, goal configurations that satisfy the robot's task requirements. The single goal configuration can be sampled using an inverse kinematic solver, e.g. TRAC-IK [Beeson and Ames 2015], that finds solutions that optimise an objective e.g. manipulability or distance from obstacles.

Instead of representing one configuration, *goal* could represent several configu-

rations. Each goal configuration represents one way that the robot can achieve its task. This approach has been used with feasible path planners to improve their success rates [Drumwright and Ng thow hing 2006, Dalibard et al 2009, Ellekilde and Petersen 2013]. This approach requires a fixed number of goal configurations to be computed before planning starts. Using a large number of goal configurations enables the planner to exploit the robot arm's redundancy, however, these configurations may take significant time to compute. Additional goal configurations may be useful for difficult planning queries, but might not be required for simple queries.

Instead of representing a constant number of configurations, *goal* could represent a *workspace goal region* [Berenson and Ferguson 2009, Berenson et al 2011]. The goal maintains a list of configurations, but has is capable of sampling more goal configurations that satisfy the robot's task. During some iterations a constant number of goal configurations are sampled and added to *goal's* list of target configurations. This means that the number of goal configurations grows as the planner takes more computation time to find a solution. Less time is spent computing inverse kinematic solutions in queries where the planner can quickly find a solution.

Bertram et al [2006] propose a variation of the RRT algorithm that does not require configuration space goals to be computed before planning, it plans toward workspace goals. Their planner constructs a tree, similar to how RRT constructs a tree but without goal biasing. The vertices added to the tree are ranked according to how close they put the robot to a workspace goal and the distance to the nearest obstacle. On some iterations, the planner selects the highest ranked vertex in the tree and attempts to extend it in a random direction. This new extension is only kept if it results in the tree becoming closer to the workspace goal. Extensions then continue in the same direction in configuration space until an extension fails. Vertices are removed from the ranked list when extensions from them have failed a specific number of times.

The Jacobian Transpose Rapidly Exploring Random Tree (JT-RRT) [Vande Weghe et al 2007] extends the approach of Bertram et al [2006] and also does not require the use of an inverse kinematics solver. Instead of randomly extending the vertex with the

highest ranking in a random direction, it is extended directly toward the workspace goal using the robot's Jacobian.

Keselman et al [2014] use two instances of the JT-RRT planner in their Forage-RRT planner. Forage-RRT constructs a course tree with one of the JT-RRT planners. The purpose of this tree is to achieve large configuration space coverage. Each vertex in this tree is added to a queue. Vertices in this queue are sorted such that the highest rank vertex is closest to a workspace goal. The second JT-RRT planner is then used to perform a fine grained search, expanding from the highest ranked vertices. The purpose of this fine-grained JT-RRT is to extend vertices from the coarse grained JT-RRT's tree into the goal. Expansions that fail to extend into the goal result in the vertex being removed from the queue of promising vertices.

Dragan et al [2011a] modified the Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [Zucker et al 2013] trajectory optimizer to be able to handle a set of goal configurations. They found that considering a set of goals improved the paths that were found by CHOMP. An extension to this work considered using a Support Vector Machine (SVM) [Boser et al 1992] to pick a good goal configuration to use with CHOMP before planning [Dragan et al 2011b]. These results suggest that specifying a goal as a set of configurations may allow asymptotically optimal planners, e.g. RRTConnect* [Akgun and Stilman 2011, Klemm et al 2015, Jordan and Perez 2013], to find better paths.

2.9 PATH PLANNING IN OTHER AGRICULTURAL ROBOTS

Two other grape vine pruning robots are under development, one by Vision Robotics [Vision Robotics Corporation. 2017] and another at the Israel Institute of Technology [Nir 2014]. There is no literature on the robot being developed by Vision Robotics, but their online video indicates that they are cutting the vine with secateurs and are using a low degree of freedom robot arm (possibly 3 degrees of freedom). It is unclear how the path planning works.

The vine pruning robot being developed at the Israel Institute of Technology relies on a human to specify cut points. Cuts are made using secateurs that are attached to the end of a six degree of freedom robot arm. A camera is used to provide feedback on the relative position of the secateurs to the cut position while individual joints on the robot arm are rotated to bring the secateurs closer to the cut position. Collision detection is not performed.

Two previous agricultural robots, one for apple harvesting [Nguyen et al 2013] and one for precision weeding [Lee et al 2014], compute collision free paths using joint space planning for high degree of freedom robot arms. The weed spraying robot uses the same six degree of freedom UR5 robot arm as the vine pruner. Before the robot is used, a database of paths is computed for the robot arm using the RRTConnect planner. This data-base is constructed assuming that there will only ever be one obstacle in the environment and that it can be encapsulated with a single fixed-size sphere directly below the robot arm (Fig. 2.12). At runtime, a path is selected from the database and it's start and end configurations are repaired to match that of the planning query. CHOMP is then used to optimise the path. This approach is not appropriate for the vine pruning robot because it relies on the environment always being the same and cannot account for the variability of vines.

The proposed apple harvesting robot uses a custom built nine degree of freedom manipulator for picking. They tested a number of feasible sampling-based path planners and found that RRTConnect had the smallest computation times. Sensing errors are not accounted for and may cause collisions when the real robot is used.

Both the proposed apple harvester (Fig. 2.13) and weed spraying robots (Fig. 2.12) only consider one robot arm configuration to reach each target, when there are actually many. Considering multiple goal configurations would likely improve plan execution times of the trajectories that result from the plans found by the planner.

Table 2.2 compares the apple harvesting, weed spraying, and vine pruning robots. We see that the vine pruning and apple harvesting robots are the most similar in task because planning will always require the arm to come very close to obstacles and the

position of these is not known prior to operation. Implementing the system used by the apple harvester could be a good start for developing a system that path plans for vine pruning.

A common solution for agricultural robots is to build a custom manipulator with low degrees of freedom to reduce self collisions. Path planning problem is then reduced to generating a trajectory for the end-effector to move to the target without collision detection [Edan et al 2000, Scarfe et al 2009, Hayashi et al 2010]. The vine pruning robot could be redesigned with two Cartesian robot arms, one on each side of the vine. This may simplify path planning but would not eliminate the need for collision detection. Using a single six degree of freedom robot arm could be faster as single motions can sweep through multiple cuts.

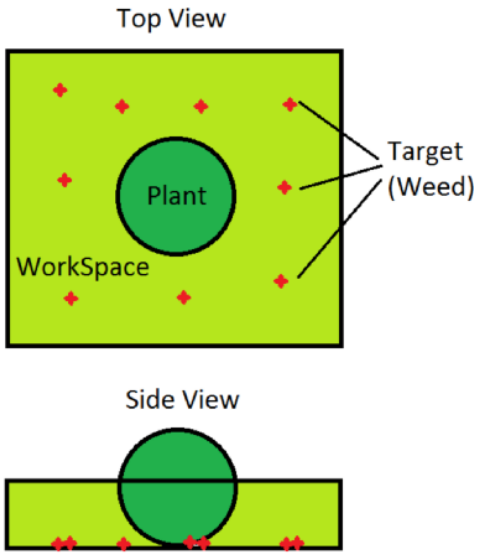


Figure 2.12: Weed spraying robot environment. Weeds (targets) lie on the ground plane and the plant (obstacle) is approximated by a single sphere [Lee et al 2014].

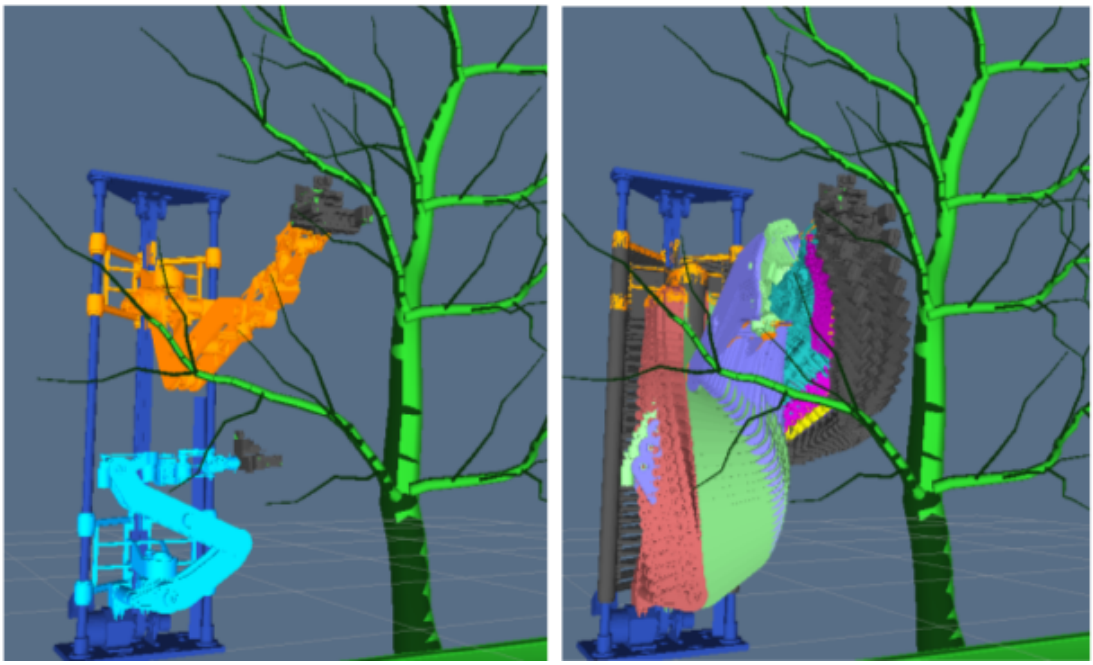


Figure 2.13: Proposed apple harvesting robot path planning scene [Nguyen et al 2014].

Table 2.2: Comparison of agricultural robots that use joint space path planning.

Robot	**Apple harvester**	**Weed sprayer**	**Vine pruner**
Robot arm	9 dof custom built	6 dof UR5	6 dof UR5
Environment obstacles	Apple tree without leaves (figure 2.13)	Plant approximated by a sphere (figure 2.12)	Grape vines, wires, and posts.
Queries	Pick apples attached to the tree	Spray weeds randomly located on ground	Cut specific vines
Obstacle positions known before operation	No	Yes	No
Account for sensing uncertainty	Yes by fitting large sphere over obstacle	No	Required

2.10 SUMMARY

Recent research into robot arms has focussed on using sampling-based motion planning. This is because sampling-based methods do not require an explicit representation of configuration space and can compute collision-free paths reasonably quickly in more than two dimensions.

Collision detection is one of the main computation time bottle-necks for sampling-based path planners. Previous research has attempted to reduce the number of collision checks using heuristics e.g. by using workspace obstacle proximity and/or lazy collision checking. In Chapter 4 I propose a fast collision detector that exploits the structure of grape vines to speed up path planning.

Path planning for robot arms becomes more difficult in the real world environment with imperfect sensor data. Appropriate adjustments to planning can be made if the distribution of errors is modelled, or a tolerance can be added to all obstacles. These methods are not appropriate for some robots that do not have error models and are required to perform tasks very close to obstacles, such as harvesting or pruning. In Chapter 4 an approach is proposed that uses an adaptive safety margin to improve the

vine pruning robot's reliability without preventing it from getting close to canes for pruning.

Quickly finding high-quality paths is an important problem in path planning as it allows robots to operate efficiently. Asymptotically optimal path planners, e.g. RRT*, can find the optimal solution, but are slow to converge. Local optimisers, e.g. short-cutting, can quickly improve an initial solution, but do not converge to the optimal solution. In Chapter 6 an approach is proposed that integrates a short-cutting local optimiser with the asymptotically optimal RRTConnect* path planner to quickly find short paths.

Robot arm tasks are typically specified in the robot's workspace, e.g. move the robot arm to a specific Cartesian position, but path planners operate in the robot's configuration space. Often only one configuration space target, out of many possible candidates, are used as the path planners goal. Previous work has looked at using all of the candidate configurations with feasible path planners to increase the chances of finding a collision free path. Little work has been performed to see how using multiple candidate configurations can effect the performance of asymptotically optimal path planners e.g. to see whether they can find shorter paths faster. In Chapter 7 effects of using multiple configuration space goals (all corresponding to the same workspace goal) on an asymptotically optimal planner are investigated.

Chapter 3

DEVELOPMENT OF A PROTOTYPE ROBOTIC PRUNING SYSTEM

The approaches proposed in this thesis were evaluated on a vine pruning robot. The approaches described in Chapters 6 and 7 were additionally evaluated on a robot for reaching into cubicles to see how they would perform on a task other than vine pruning. Both experiments make use of the six degree of freedom Universal Robot (UR) UR5 robot arm.

3.1 UR5 ROBOT ARM

The UR5 robot arm has six rotational joints that are capable of making two full rotations. The robot arm with its joints labelled is shown in Fig. 3.1 and its specifications can be found in Appendix. 8.2. I used the open source Universal Robot drivers[1]. Some of the joints of the UR5 robot arm had to be limited to account for un-avoidable collisions between the robot arm and itself, or the robot arm and the wall it was mounted on.

3.1.1 Limiting joints

The elbow joint of the UR5 robot arm used in our experiments was limited to the range $[-\pi, \pi)$. This is because the arm has a self collision when the elbow joint is close to $\pm\pi$ as shown in Fig. 3.2. This self collision causes the UR5's configuration space to

[1] `https://github.com/ros-industrial/universal_robot`

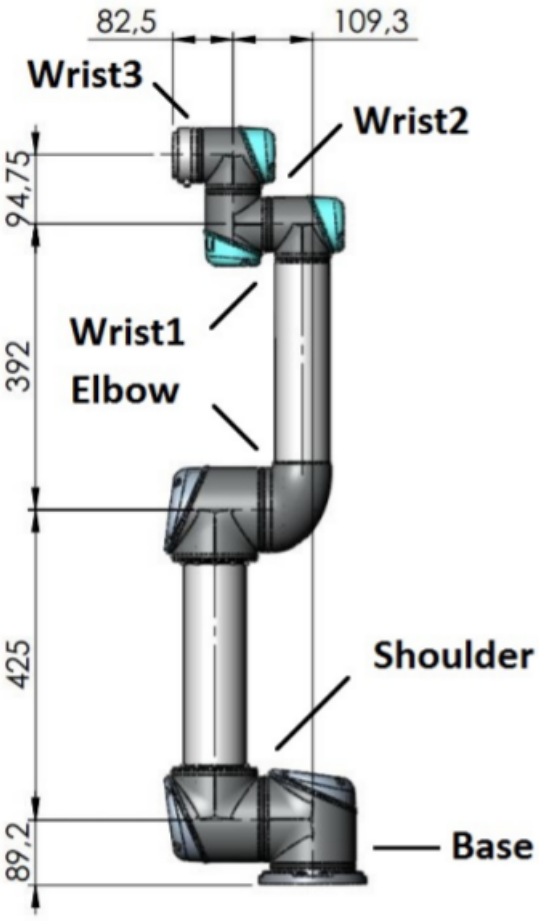
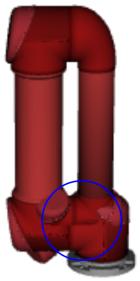


Figure 3.1: UR5 robot arm with labelled joints and dimensions in mm. Image courtesy of de Gier [2009].

be split into three disjoint sets depending on whether the elbow joint is in $[-2\pi, -\pi)$, $[\pi, \pi)$ or $[\pi, 2\pi)$. Since these sets are disjoint, it is not possible to find a collision-free path where the start and goal positions of the elbow joint are in different sets. This has been breaking the path planning with the UR5 for some time and had been thought to be a software bug. I have reported this issue to the Universal Robot repository on GitHub.

I additionally limited the range of the shoulder lift joint to $[-\pi, 0)$ in our experiments with the vine pruning robot. This is because the vine pruning robot has a back wall, which further splits the UR5's configuration space as shown in Fig. 3.3.



(a) UR5 in self collision. The self collision is circled in blue.

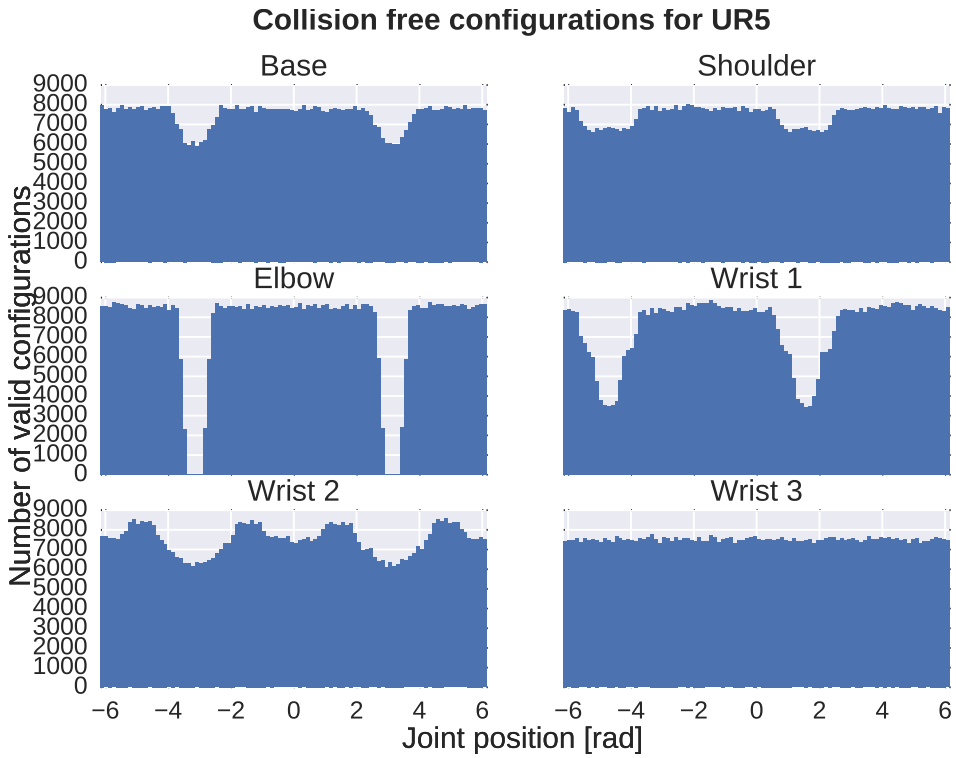
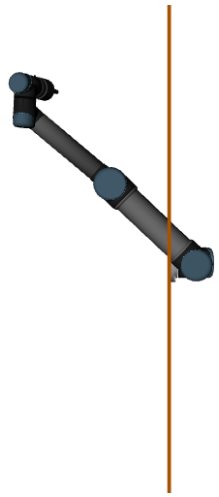


Figure 3.2: Individual joint positions for collision-free configurations from one million randomly sampled configurations. There were no valid configurations when the elbow joint is close to $\pm\pi$ regardless of the positions of the other joints. This makes the UR5's configuration space disjoint depending on the position of the elbow joint.



(a) UR5 in collision with mounting wall.

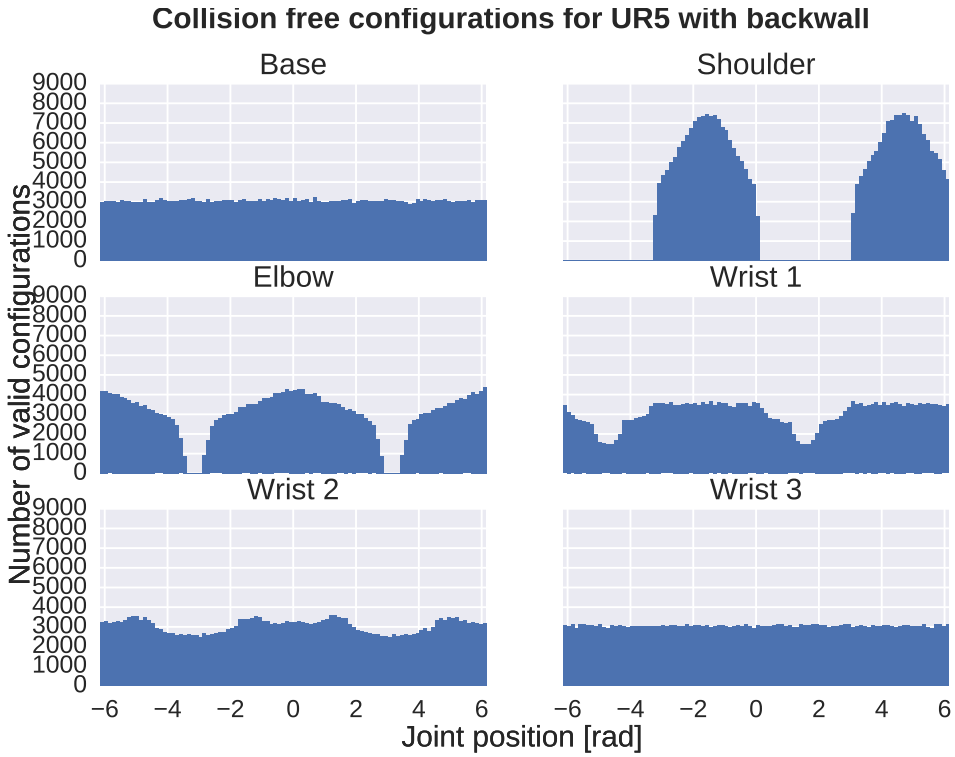


Figure 3.3: Mounting the UR5 on a flat surface (brown) causes its configuration space to be disjoint depending on the position of the shoulder lift joint.

3.2 ROBOT OPERATING SYSTEM AND MOVEIT SETUP

The Robot Operating System (ROS) [Quigley et al 2009] and Moveit [Chitta et al 2012] middleware libraries were used for both the vine pruning and cubicle picking robots. ROS and Moveit use a distributed architecture. At runtime various parts of the software are executed asynchronously in different *nodes*. Nodes are stand-alone executables. They may communicate with each other at runtime by publishing or subscribing to topics.

The main benefits of using these libraries were the RVIZ visualisation library and the open source Universal Robot drivers. The vine pruning robot setup displayed in RVIZ is shown in Fig. 3.4. Moveit provides full path planning functionality using the Open Motion Planning Library (OMPL) [Sucan et al 2012] and collision checking with the Flexible Collision Library (FCL) [Pan et al 2012]. To enable faster and more effective path planning a new collision detector (see Chapter 4) and path planners were developed.

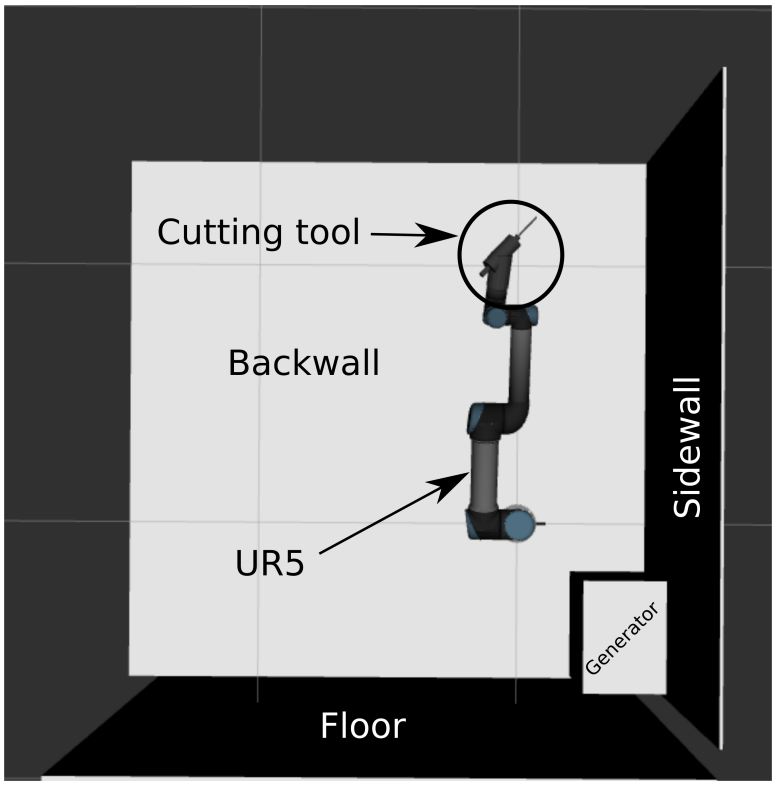


Figure 3.4: Annotated image of vine pruning robot setup in ROS's RVIZ visualisation software.

3.3 VINE PRUNING ROBOT

The vine pruning robot straddles over the row of grape vines it is pruning (Fig. 3.5). This hub blocks out sunlight so that lighting can be controlled using LEDs. While the robot is being pushed along the row of vines, it uses stereo cameras to capture images of the vines. These images are used by a 3D reconstruction algorithm [Botterill et al 2013] to build a model of the plants.

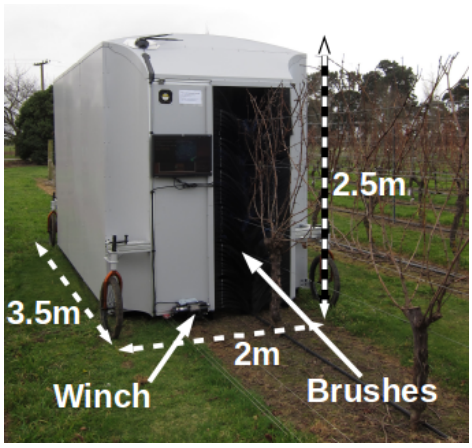
An Artificial Intelligence (AI) algorithm [Corbett davies et al 2012] is used to determine which two canes on the plant should be spurred (cut to a length of 10-20cm) and which two canes should be left uncut with the remaining canes to be cut to about 5cm in length. A cost function was learned using expert pruner input. The AI algorithm used a brute force search to determine which combinations of canes should be left, cut short or spurred. This algorithm then output 'ideal' cut positions for canes on the vine, although it does not account for whether the robot arm can safely make

the cuts.

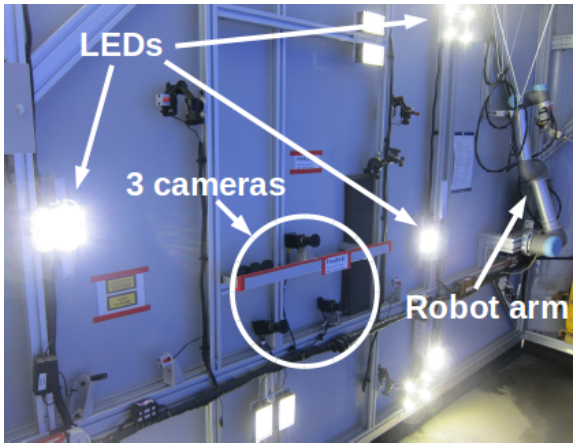
The robot arm (Fig. 3.6) has a mill-end cutting bit mounted that is used to cut the vines (Fig. 3.7). The mill end is powered with a 100 W 24,000 revolution-per-minute Maxon brushless DC motor.

The hub is stopped when the next plant to prune is located in front of the robot arm. The cut positions are then modified to positions where the robot arm can make cuts without colliding with itself or the rest of the plant (Sec. 3.3.2). A collision free path for the robot arm to reach each cut is computed using the path planner, and executed on the UR5 robot arm (Fig. 3.8). When the arm reaches each cut-point it swipes through it with the router (Fig. 3.9). If this is successful the vine will be cut (Fig. 3.10).

The router was initially used instead of secateurs because it requires one less degree of freedom to be specified at the cut point (secateurs need to be orientated such that the cane fits between the cutting edges). However, the router needs to be swiped in a direction near perpendicular to canes in order to make a cut. This direction requirement offsets the benefit of using a router instead of secateurs.



(a) Hub for the vine pruning robot.



(b) Inside the vine pruning robot hub.

Figure 3.5: Vine pruning robot hub (a) and inside the hub (b) [Botterill et al 2016].

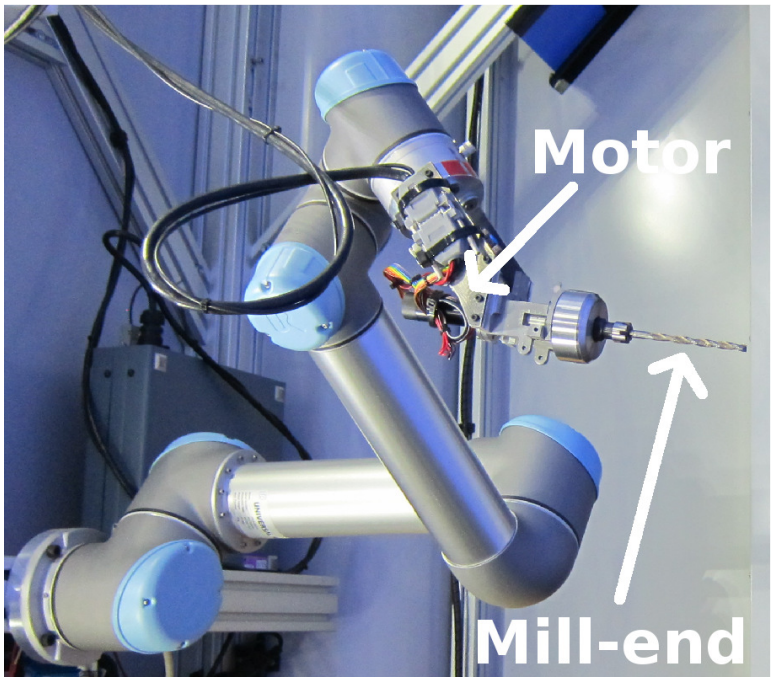


Figure 3.6: Labelled robot arm [Botterill et al 2016].

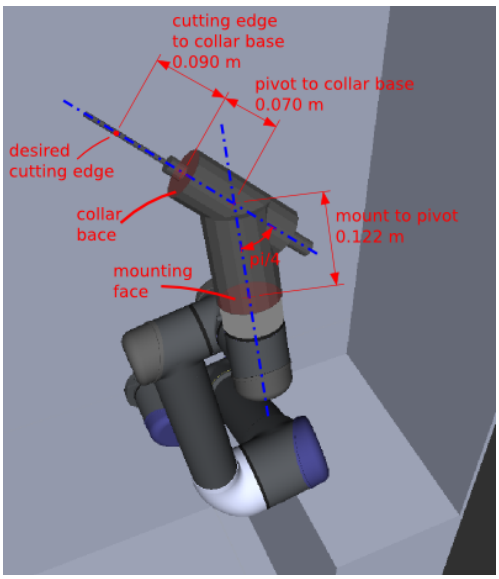


Figure 3.7: Cutting tool dimensions.

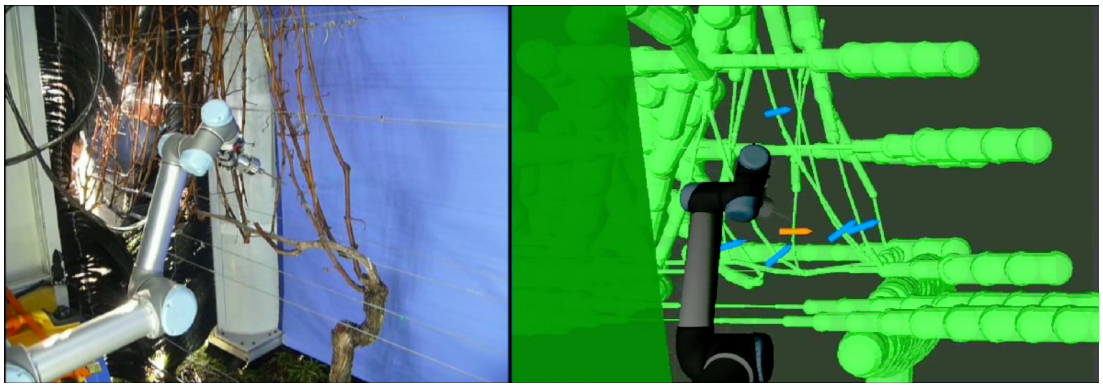
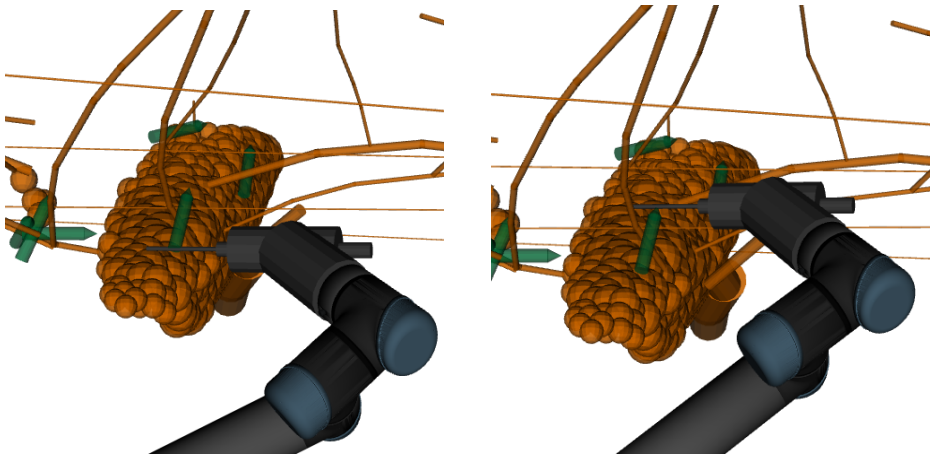


Figure 3.8: Robot arm reaching to make the last of six cuts on a plant. The vine on the right has had an adaptive safety margin applied [Botterill et al 2016].



(a) Robot arm at start of swipe motion. (b) Robot arm at end of swipe motion.

Figure 3.9: Robot arm at the start and end of a cut swipe motion

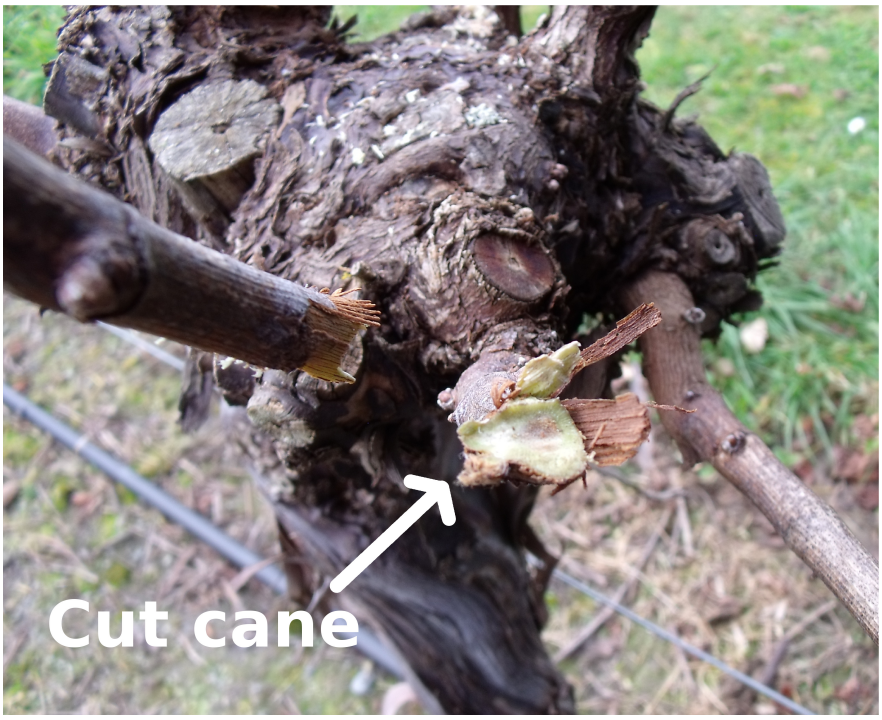


Figure 3.10: Separation cut [Botterill et al 2016].

3.3.1 Software architecture

The path planning software was designed to run on data streamed from either the online 3D reconstruction algorithm, or saved data as shown in Fig. 3.12. The paths taken by the software when the reconstruction software was being run (e.g. during a field test) are shown in blue. The paths taken by the software when saved 3D reconstruction data was being used are shown in green.

The 3D reconstruction software streamed a model of the environment (see Appendix 8.2 for some sample data for one plant). The canes and trunk were represented as 3D polylines (lines that pass through a series of way points) and the plant's head was represented with spheres (Fig. 3.13). The cutpoints found by the AI algorithm were represented by the Global Unique IDentifier (GUID) of the cane they were on and a distance from the start of the cane. Knowing what cane a cut was on was important for computing swipe motions. All of this data was represented in a reference frame located at the start of the row.

The cut and plant position data had to be transformed into the reference frame

Figure 3.11: Reference frames for the cameras and planning software. The reference frame used by the Universal Robot controller is located in the same position as the one used by the planning software but rotated 90 degrees counter clockwise.

used by the path planning from the camera reference frame that is used by the 3D reconstruction software as shown in Fig. 3.11. The 4x4 transformation matrix from the camera frame to the planning frame was calculated with the use of a marker attached to the end of the robot arm. The 3D reconstruction software was used to compute the pose of the marker relative to the camera frame, and the robot arm's forward kinematics routine was used to compute the pose of the marker relative to the planning reference frame. Knowing the pose of the marker relative to both of these reference frames the transform from the camera frame to the planning frame was computed.

Points output from the 3D reconstruction software were transformed into the path planning reference frame as follows:

$$X_{\text{planning}} = T_{\text{camera\_to\_planning}} X_{\text{camera}} \tag{3.1}$$

Where X_{planning} represents the coordinates of a point in the path planning reference frame, X_{global} represents the coordinates of a point output by the 3D reconstruction software and $T_{\text{camera\_to\_planning}}$ is the 4x4 transformation matrix from the camera refer-

ence frame to the path planning reference frame.

The path planning module was isolated from the input data source (online 3D reconstruction software or saved data) by a First In First Out (FIFO) named pipe. When the 3D reconstruction software was running the vines and cutpoints were passed on to the named pipe. If saved data was being used it was read directly from file as if a named pipe was being used. This meant that the path planning module had the same interfaces regardless of whether the robot was being used in the field or being tested in simulation.

The UR5 robot arm drivers were isolated from the robot arm by a Moveit node. When the physical robot was being used (e.g. during field tests) the robot drivers would control the physical robot arm. When tests were being performed in the lab the drivers would communicate with a simulated version of the UR5 provided by Universal Robot (Fig. 3.14).

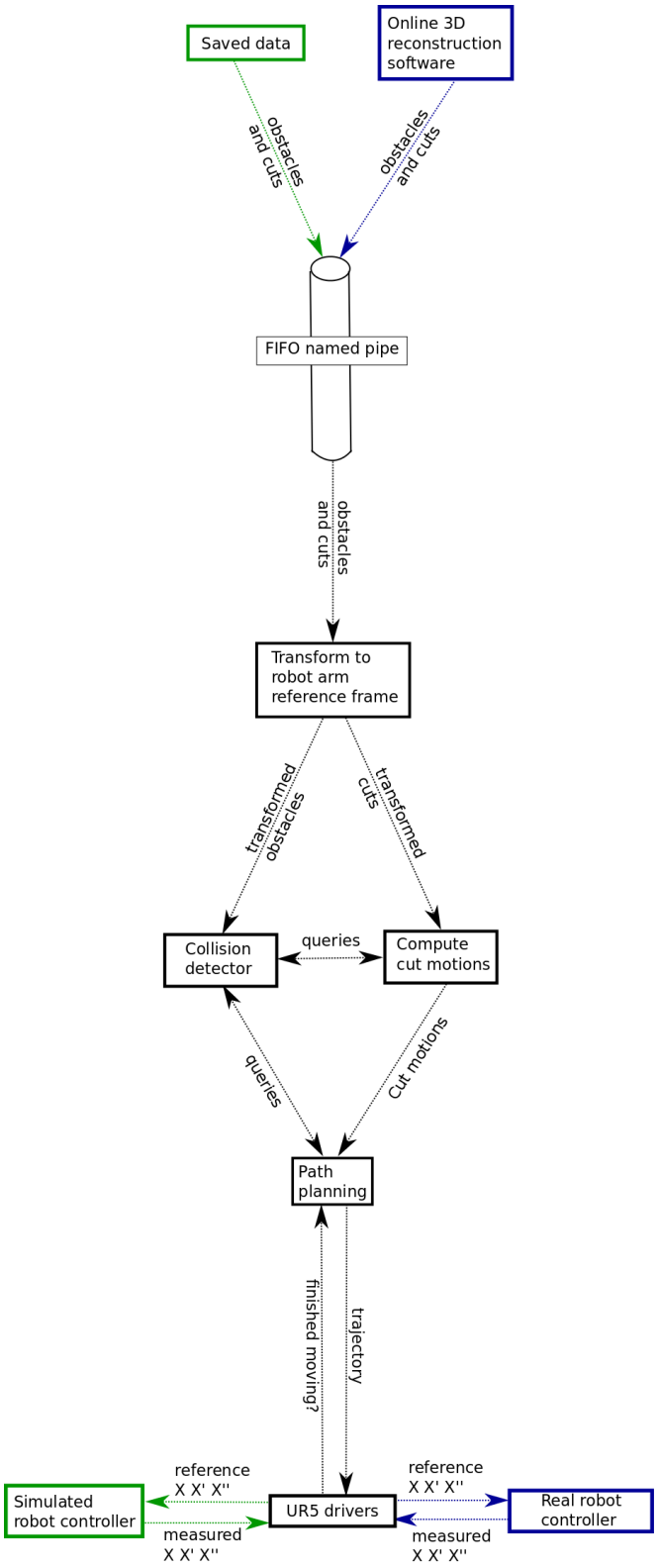
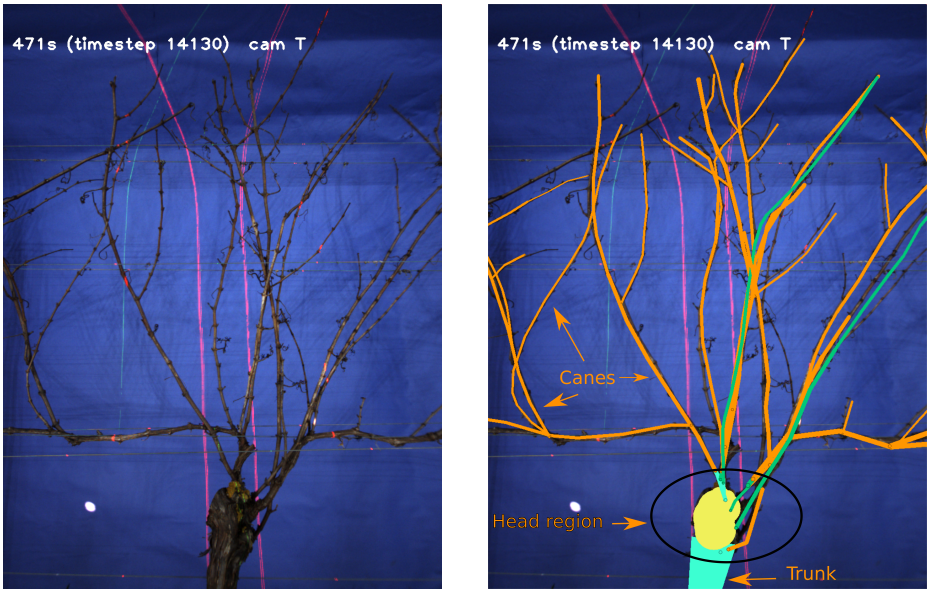


Figure 3.12: Information flow for vine pruning robot. Blue paths are only taken when the physical robot is being used. Green paths are only taken during simulation. X, X', X'' represent the positions, velocities and accelerations of the joints on the robot arm.



(a) Raw image of a vine. (b) Annotated image of a vine.

Figure 3.13: Example of a raw vine image before 3D reconstruction has been performed (a) and this image annotated with the cane, trunk and head features found by the reconstruction software labelled (b).

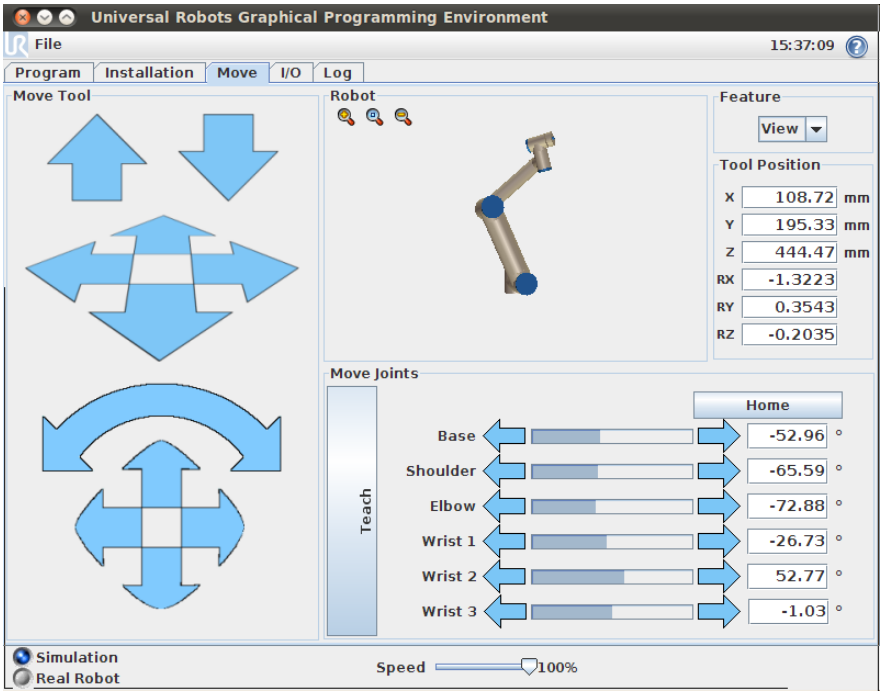


Figure 3.14: UR5 robot simulator.

3.3.2 Computing cut motions

To make a cut the robot arm had to swipe a spinning mill end through the vine. The swipe motion had to start 4cm before the cane and end 6cm after as shown in Fig. 3.15. The 4cm before the vine was to account for some reconstruction error in the model of the vine. The 6cm after the cut position was to make sure a separation cut was performed, using shorter distances often resulted in the cane bending out of the way of the router without a cut being performed. To ensure that the mill could separate the cane at the cutpoint the direction of the swiping motion, the cane and the mill end all had to be perpendicular to each other.

The best places to make each cut were computed using AI algorithm [Corbett davies et al 2012]. This algorithm did not account for collisions between the robot arm and itself, or collisions between the robot arm and the environment. This meant that it was not always possible to cut the vines at positions specified by this algorithm using the UR5 robot arm.

The procedure for computing a swipe motion is outlined in Fig. 3.16. A tool direction perpendicular to the cane's direction is sampled with a bias to directions away from the robot's base. A swipe direction perpendicular to both the cane and tool directions is then calculated. The SampleSwipeConfigs function then calculates configurations for the start middle and end points of the swipe motion using an inverse kinematic solver. The middle configuration is calculated first and then the start and end configurations are computed considering the swipe direction and its length (Fig. 3.15). If the path found by interpolating between start, middle and end is collision free (ignoring collisions between the cutting tool and cane to be cut) it is returned, otherwise failure is returned.

If a collision free swipe motion cannot be calculated for a cut position specified by the AI algorithm then a new cut position on the cane and cut direction are sampled on the cane 0.06mm further from the trunk of the plant. If no swipe motions can be found after 5000 attempts then the cane will not be pruned. Moving the cut position down

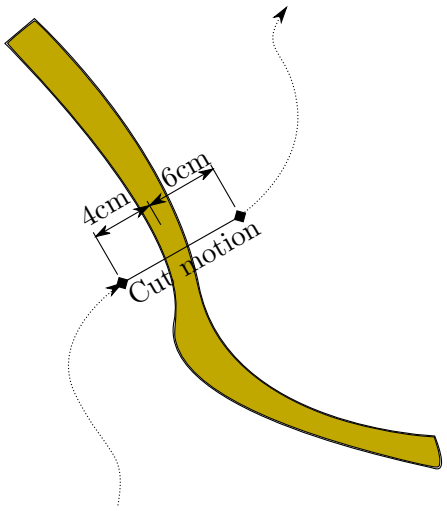


Figure 3.15: Path planning to cut a vine through with a router. The dashed lines represent unconstrained path plans. The solid line shows the 10cm swipe motion that is computed before any path planning is performed.

```
1: function ComputeSwipeMotion(cane_dir, position)
2:     tool_dir ← Randomly sampled tool direction perpendicular to cane_dir
3:     swipe_dir ← Direction perpendicular to cane_dir
4:     start, middle, end ← SampleSwipeConfigs(position, cane_dir, tool_dir,
   swipe_dir)
5:     swipe_motion ← Path made by interpolating between start, middle and end
6:     // Ignore collisions between cutting tool and cane being cut
7:     if swipe_motion is collision free then
8:         return swipe_motion
9:     else
10:        return Failure
11:    end if
12: end function
```

Figure 3.16: Procedure to attempt to calculate a swipe motion to make a cut.

the cane ensures that the algorithm does not always fail to find swipe motions when none are possible using the initial cut position. The algorithm was limited to 5000 iterations in the interest of limiting computation time. The cut positions were moved by 0.06mm after each unsuccessful iteration to ensure that cuts were never made more than 30cm from their ideal position. On average 82% of cuts could have swipe motions found for them. Fig. 3.17 shows how far the cut position had to be moved from the position supplied by the AI algorithm to get a collision free cut motion.

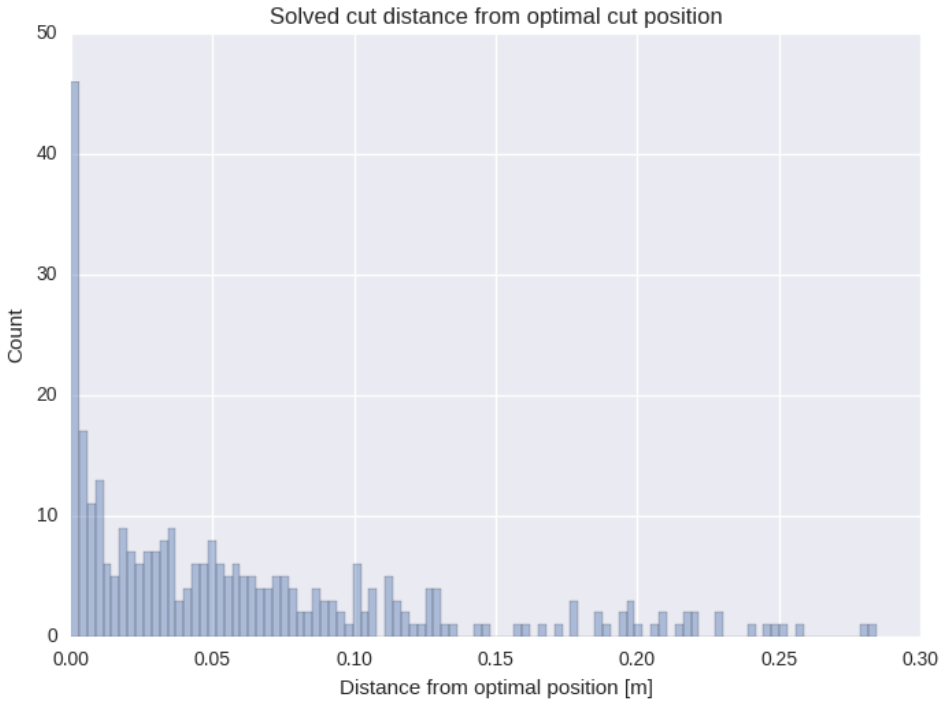


Figure 3.17: Distance from optimal cut point for solved cuts obtained in simulation from data captured on a row of Sauvignon Blanc at Lincoln University.

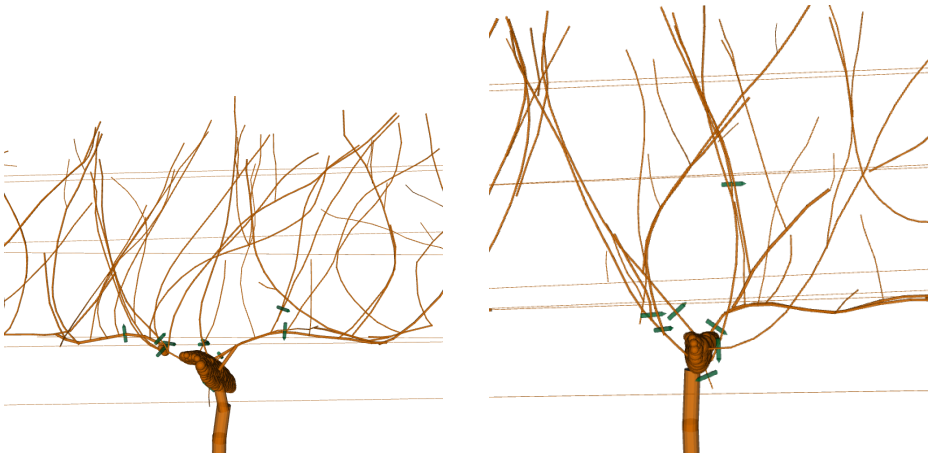
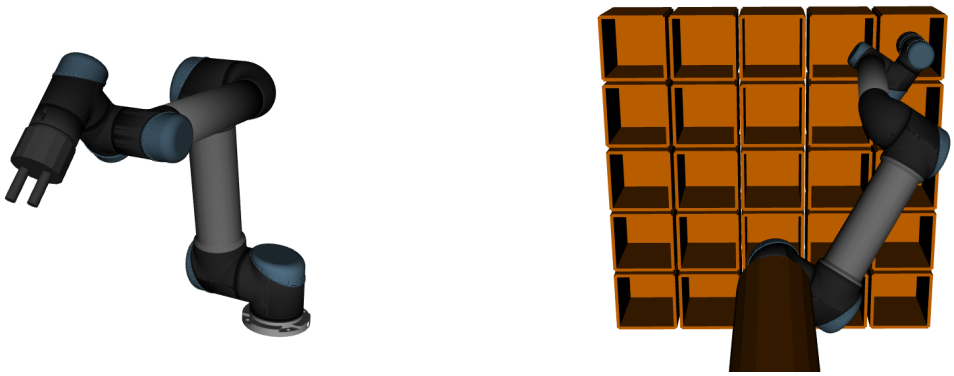


Figure 3.18: Two reconstructed grape vines with modified cut motions shown in green.

3.4 CUBICLE PICKING ROBOT

The cubicle picking environment (Fig. 3.19) was designed to be similar to that used in previous research [Choudhury et al 2016a, Phillips et al 2012, Ratliff et al 2009] and the 2015 Amazon Picking Challenge [Correll et al 2016]. The robot should reach into each cubicle without colliding with it. The dimensions for the environment are shown in Fig. 3.20. These dimensions were selected so that the UR5 could reach into every cubicle with more than one configuration.

The dimensions for the robot's gripper are shown in Fig. 3.21. These dimensions were selected to approximate a small gripper that could be attached to a robot arm for picking up small objects.



(a) Robot arm with gripper model.

(b) Robot arm reaching into a cubicle.

Figure 3.19: Cubicle picking scenario.

The planner had to compute plans so that the robot arm would reach from its start position in one cubicle into another. Exiting the start cubicle and entering the goal cubicle both required fine motion plans. The Flexible Collision Library (FCL) [Pan et al 2012] was used for collision detection. An analytical IK solver for the UR5 [Hawkins 2013] was used to generate the eight robot arm poses to reach the arm into the centre of each cubicle with the grippers parallel to the bottom of the cubicle.

3.4.1 Software architecture

The information flow for the cubicle picking robot software is shown in Fig. 3.22. The model of the cubicles was specified to match that in Fig. 3.20. An analytical IK solver for the UR5 was used to compute configurations that put the gripper into the middle of the cubicle at a specified orientation. The collision detector used information about the location of cubicle walls and a model of the UR5 to determine whether robot arm configurations were part of C_{free} or C_{obs}. The path planner computed path plans to the target configurations supplied by the IK solver using the collision detector. These paths were then forwarded to the UR5 robot arm drivers.

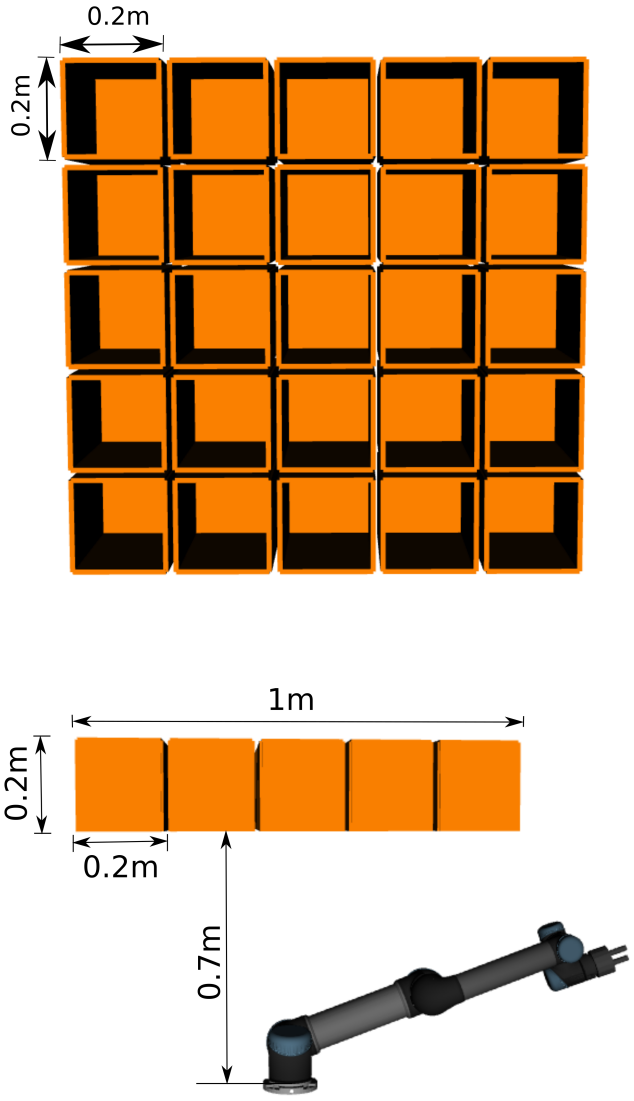


Figure 3.20: Dimensions for cubicles experiment setup.

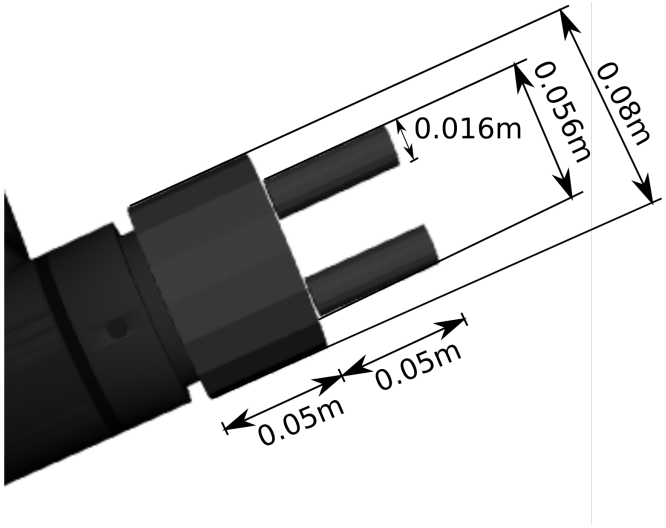


Figure 3.21: Gripper for cubicles experiment with dimensions.

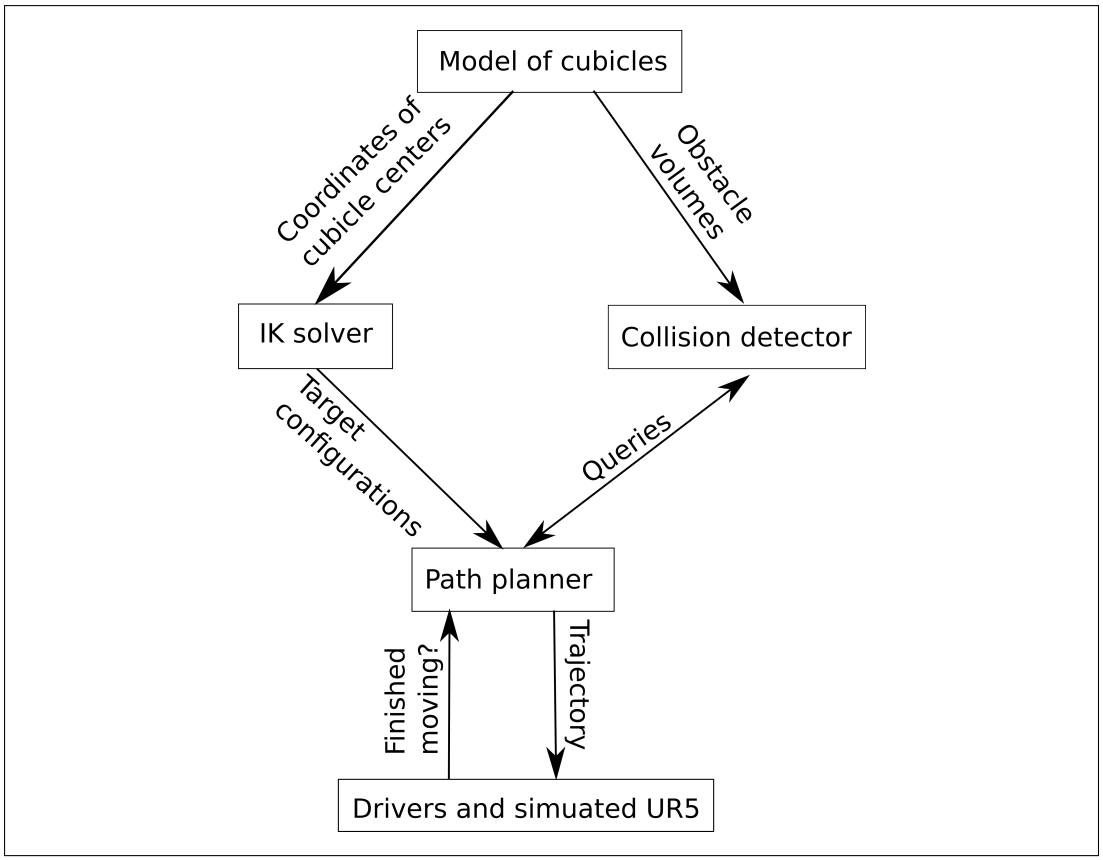


Figure 3.22: Information flow for the simulated cubicles picking robot.

Chapter 4

A SPECIALISED COLLISION DETECTOR FOR GRAPE VINES

4.1 INTRODUCTION

Efficient online motion planning is important so that the vine pruning robot can prune quickly. Path planning algorithms are often slow when they are used with an inefficient collision detector. This chapter details an efficient collision detector that was specifically designed for detecting collisions between a robot arm and a grape vine. It models vines using capsules and spheres which provide fast intersection tests. Spatial partitioning is performed using a one dimensional sweep and prune algorithm. Objects are bounded by spheres to further improve computation times. This specialised collision detector is tested on the vine pruning robot, and its performance is compared to the popular Flexible Collision Library.

It is common for more than 90% of path planning computation time to be spent in collision detection routines [Sánchez and Latombe 2003a]. These times can be lowered by reducing the number of collision detection calls required, or speeding up the collision detector. Previous work focusses on reducing the number of times the collision detector is called by specialising the path planner to the robot's configuration space [Bohlin and Kavraki 2001, Kuffner and Lavalle 2000]. This is difficult to do with high degree of freedom robot arms because the configuration space cannot easily be visualised, however, it is possible to visualise the robot's environment. This means that a collision detector can be specialised for the task of vine pruning more easily than a state of the art general path planner.

Path planners use collision detectors in two ways, to determine whether a specific robot configuration would result in workspace collisions, and to determine whether a straight-line path in the robot's configuration space (a motion) would lead to a workspace collision. Configuration space motions can be collision checked by sweeping workspace volumes and looking for intersections between the volume and any obstacles (continuous motion validation), or by collision checking discrete points in the configuration space motion (discrete motion validation, see Fig. 4.1). Discrete motion validation is often used because it does not require complex swept-volume computations. Discrete motion validation is used in this thesis, which means that the collision detector does not need to compute swept volumes for configuration space motions.

4.2 EXISTING APPROACHES TO COLLISION DETECTION

The collision detector must keep a model of the robot and its environment. Objects are commonly represented as a polygon soup (e.g. from a mesh), a solid primitive (e.g. sphere, cylinder) or a combination of primitives. Ideally an object's representation will geometrically be a good fit and allow for fast collision checking.

A naive collision detector would perform all pairwise collision checks between objects in the environment. For the robot it would check every object in the robot arm against every object in the environment. This is slow when there is a large number of objects in the environment or robot arm. Most of the pairwise checks can be avoided by keeping objects in data-structures that allow objects within specific subregions to be accessed efficiently. Two approaches are to store objects with bounding volumes or in spatial partitions.

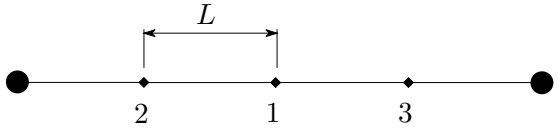


Figure 4.1: Discrete motion validation with at most L distance between collision checked points. The segment is recursively bisected and the midpoints are checked for collisions in the order shown. The algorithm returns that the motion is invalid as soon as one point is found to be in collision, or valid if all points are not in collision.

The collision detector may encapsulate each individual object with a bounding volume, or each bounding volume could encapsulate multiple objects. When encapsulating multiple objects the bounding volume can reduce the number of pairwise checks. Objects that are expensive to collision check can also be encapsulated, e.g. complex polygon soups, with objects that are quick to check like spheres. It is favourable to use bounding volumes that are a tight fit and provide inexpensive intersection tests, however, there is often a trade-off between the two. Some common bounding volumes (in increasing order of complexity) are sphere, axis aligned bounding box [Bergen 1997], oriented bounding box [Gottschalk et al 1996], discrete oriented polytope [Klosowski et al 1998] and convex hull. Bounding volume approaches can be sped up by performing computation on the GPU [Pan and Manocha 2012] [Lauterbach et al 2010].

Spatial partitioning methods divide the space into regions and test whether objects overlap in the same region of space [Ericson 2004]. Three common methods are grids, trees and sweep and prune [Cohen et al 1995].

Grid methods overlay space with cells. Pairwise checks are performed between the robot and objects in the same cells. The performance of these methods is highly dependant on the resolution of the grid. If it is too fine then many objects will be members of multiple cells. If the resolution is too course then there will be many objects in each cell. Both cases lower the performance of grids [Ericson 2004].

Tree methods recursively divide the space into cells. This is similar to the grid structure, however, each cell is further split into cells until a certain tree depth or resolution on the leaf node is reached. Octrees [Meagher 1982] are a common method. They recursively divide the space into eight cubes. This continues until a maximum tree depth is reached or the leaf node volumes are smaller than a specified value.

Grids and trees work well when environment objects can be tightly bounded by cubes. Long and thin objects, such as cylinders, are not tightly bounded because all dimensions of the cube scale with the largest dimension of the cylinder, its length. Cylinders will be loosely bounded by one cube, or occupy multiple partitions. Both of these cases cause a reduction in performance [Ericson 2004].

Instead of grouping objects by the cubes they occupy, they can be maintained in a spatially sorted list and sweep and prune [David Baraff 1992, Cohen et al 1995] can be used. The minimum and maximum distances between each object and a reference point, e.g. the base of the robot, are computed and stored. This provides a range of distances that each object occupies from the base of the robot arm. If there is no overlap between two objects with these distances then they cannot be in collision. Sweep and prune finds a set of object pairs that cannot possibly be in collision based on these distances, as shown in Fig. 4.4. Intersection tests are then performed on the remaining pairs.

Information about previous collision queries can be used to speed up collision detection. This can be done by constructing trust regions where the robot is known not to be in collision [Bialkowski et al 2016], keeping track of configurations that are in collision and using them to predict whether new configurations will also be in collision based on a nearest neighbour search [Pan and Manocha 2016] or replacing the collision detector with a Support Vector Machine [Boser et al 1992] at runtime [Pan and Manocha 2015]. A very recent article has even proposed using deep learning [Lecun et al 2015] to replace the path planning and collision detection pipeline [Long et al 2016]. To work effectively these approaches require information from previous planning queries or offline training. This would limit their effectiveness for the vine pruning task where offline training cannot be performed due to real-time constraints and few planning queries are made per plant. Previous training data is invalidated when the plant is changed because the robot's C_{free} and C_{obs} change.

Safety margins can be applied to objects in the collision detector to account for sensor error. This allows a path planner to find paths with a guaranteed minimum obstacle clearance without requiring the collision detector to perform expensive distance-to-nearest-obstacle queries. These margins are often applied uniformly to all objects in the scene. The amount of safety margin required for vine pruning depends on the amount of error that is produced by the 3D reconstruction software, which is currently not known.

4.3 SPECIALISING A COLLISION DETECTOR FOR GRAPE VINES

A grape vine is made of canes and a model of the head. The canes are modelled with capsules (cylinders with hemispherical end caps) and the head is modelled with a number of spheres, as shown in Fig. 4.3. These primitives were chosen because they are quick to perform intersection tests on and are a good fit for the problem. The capsules are approximated as the union of a sphere and a ray (see Fig. 4.2) to reduce intersection testing time.

Each capsule is bounded by a sphere because sphere-sphere collision checks are faster than capsule-capsule checks. The entire model of the head labelled in Fig. 4.3 is bounded by one sphere to reduce the number of pair-wise collision checks. The robot frame and generator are modelled with planes.

A one dimensional sweep and prune algorithm similar to that in I-Collide [Cohen et al 1995] is used to further reduce the number of intersection tests that need to be performed. The minimum and maximum distances to the base of the robot are computed for each object. Collision detection is only performed on pairs of objects that have overlapping minimum and maximum distances to the base of the robot, see Fig. 4.4. The full collision detection routines are shown in Fig. 4.5 and Fig. 4.6.

To make a cut the robot arm needs to move very close to the cane. This means that any safety margins applied to the cane need to be small near parts of the robot arm when it is in a cutting configuration, but can be large further away. An adaptive safety margin is added to each object based on the minimum distance between it and any part of the robot arm when the arm is in a cutting configuration, as shown in Fig. 4.7 and Fig. 4.3. Larger margins are added to objects further from the arm.

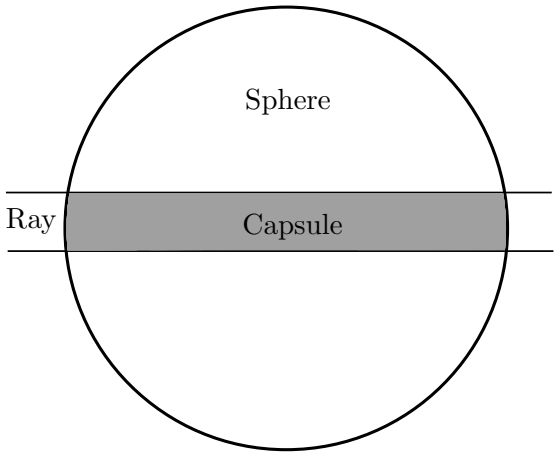


Figure 4.2: Cross-sectional view of a capsule approximated by a sphere and ray. This is a slight over-estimation of an actual capsule but allows fast intersection testing.

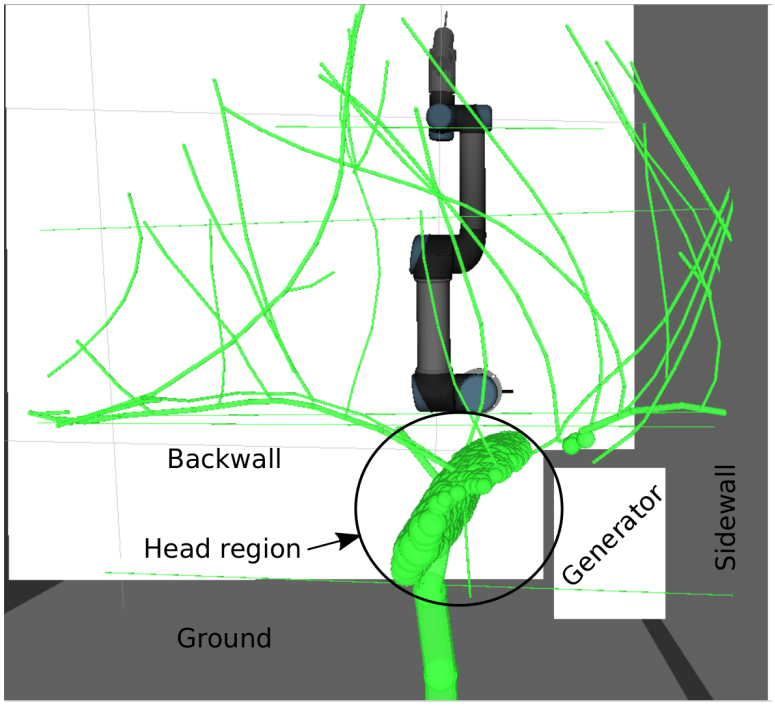
4.4 RESULTS

The specialised collision detector is compared to the Flexible Collision Library (FCL) [Pan et al 2012] through Moveit [Chitta et al 2012] using the RRTConnect [Kuffner and Lavalle 2000] planner implementation from the Open Motion Planning Library [Sucan et al 2012]. FCL was configured by Moveit to use axis aligned bounding volumes, as shown in Tab. 4.1. The robot arm was represented with a mesh provided by the Universal Robot package.

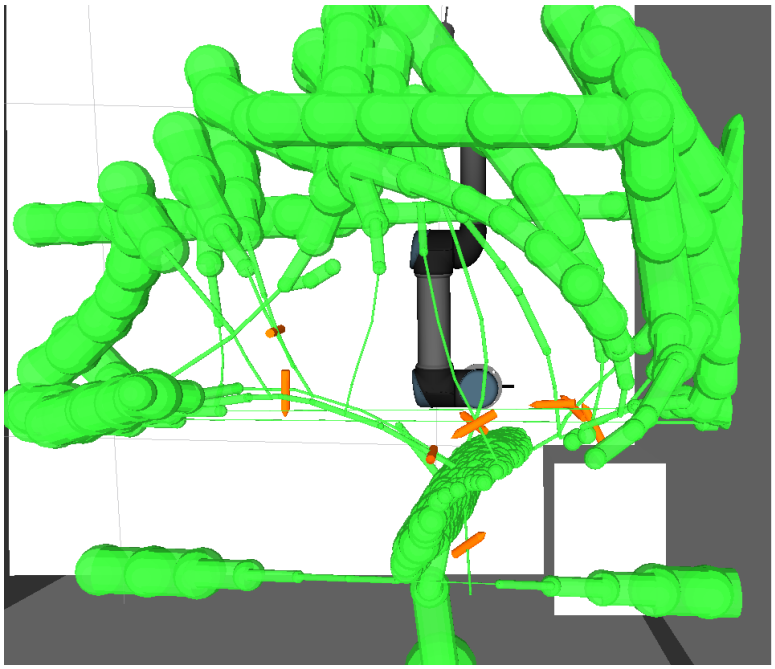
Table 4.1: Configurations of specialised collision detector and FCL

Collision detector	Arm model	Spatial partitioning	Bounding volumes
Specialised	Capsules	Sweep and prune	Spheres
FCL	Mesh	-	Axis aligned bounding boxes

The specialised collision detector has significant speed-up over FCL for full collision checks, as shown in Tab. 4.2, and for self collision checks as shown in Tab. 4.3. The collision detection time for both collision detectors was measured when checking random configurations, and then when checking only valid configurations in a second trial. The collision detector cannot terminate early when checking valid configurations meaning it can take longer. In both trials the mean times were calculated by dividing the time it took for the collision detector to check a large number of samples by the number of samples.



(a) Grape vine without safety margins applied. The specialised collision detector bound the entire head region with one sphere.



(b) Grape vine with safety margins. Cuts are marked in orange. Cuts are made using specific robot configurations. The safety margin is computed using the minimum distances between parts of the robot arm in these states and the parts of the vine.

Figure 4.3: Adaptive safety margin applied to a vine.

```
1: function SweepAndPrune(a, b)    // Determine whether a pair of objects
   may intersect
2:     if (a.min() > b.max()) or (b.max() < a.min()) then
3:         return False    // Objects definitely not intersecting
4:     else
5:         return True    // Objects could be intersecting
6:     end if
7: end function
```

Figure 4.4: One dimensional sweep and prune. min() and max() provide the minimum and maximum distances between an object and a reference point e.g. the base of the robot. If there is no overlap of min and max values of the two objects then the pair cannot intersect. Returns true if the two objects could be intersecting and false if they are definitely not intersecting.

```
1: function ObjectCollidesWithObstacles(object, obstacles)
2:     for each obstacle ∈ obstacles do
3:         if SweepAndPrune(object, obstacle) then
4:             if Bounding volumes of object and obstacle intersect then
5:                 if Intersects(object, obstacle) then
6:                     return True    // object intersects at least one obstacle.
7:                 end if
8:             end if
9:         end if
10:    end for
11:    return False    // object intersects none of the obstacles
12: end function
```

Figure 4.5: Full collision checking of an object with the rest of the scene. Returns true if *object* collides with any of the objects in *obstacles*. Intersects(a, b) performs explicit intersection testing between a and b and returns true if they are intersecting.

```
1: function InCollision(configuration, environment)
2:     robot_arm ← GetTransformedRobot(configuration)
3:     for each part ∈ robot_arm do
4:         adjacent_parts ← part ∪ all parts in robot_arm joined to part
5:         obstacles ← scene ∪ robot_arm \ adjacent_parts
6:         if ObjectCollidesWithObstacles(part, obstacles) then
7:             return True     // There is at least one collision
8:         end if
9:     end for
10:    return False    // This robot configuration is collision free.
11: end function
```

Figure 4.6: Full collision checking of the robot arm with itself and the environment. GetTransformedRobot(configuration) computes the workspace model of the robot arm in *configuration*.

Table 4.2: Mean times for full collision checking with smart safety margin

Collision detector	Random state [s]	Valid state [s]
Specialised	3.0×10^{-6}	5.9×10^{-6}
FCL	1.5×10^{-4}	2.5×10^{-4}
Improvement	50×	42×

Tab. 4.4 shows that the sweep and prune algorithm saves over 90% of pairwise object checks between the robot arm and the vine, and that only 0.23% of object pairs required explicit intersection testing. Pairwise collision checks now only take a small portion of total collision detection time, as shown in Fig. 4.8. 59% of the computation time is spent computing transforms for the robot's pose from the input joint angles (33%) and applying these transforms to a robot model (26%).

Using the specialised collision detector (SCD) instead of FCL resulted in the speed up in path planning times shown in Fig. 4.9. The path planner was tested by calculating all of the paths required to prune 37 plants twice each, which is approximately 650 paths. Different paths were generated each time a specific plant is pruned due to

Table 4.3: Mean times for self collision checking with smart safety margin

Collision detector	Random state [s]	Valid state [s]
Specialised	2.8×10^{-6}	2.9×10^{-6}
FCL	6.0×10^{-5}	5.0×10^{-5}
Improvement	21×	17×

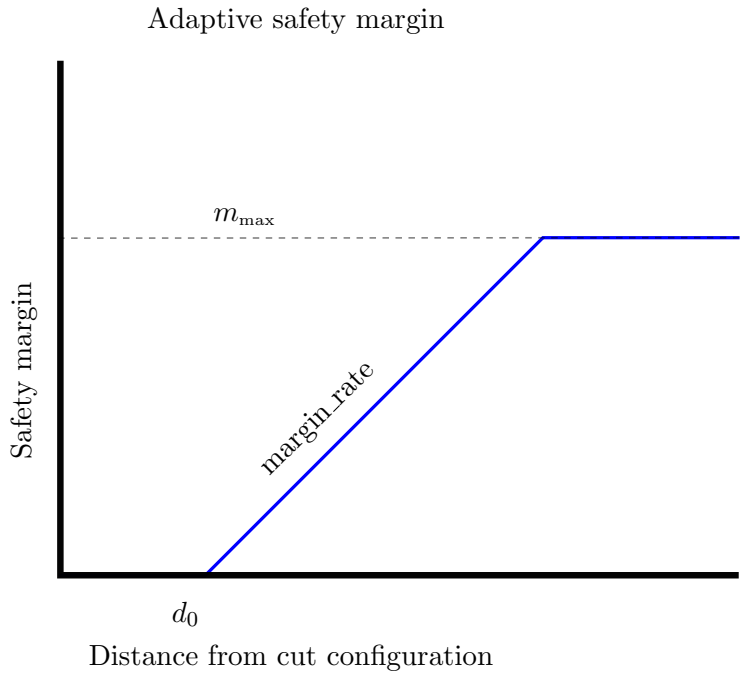


Figure 4.7: The safety margin applied to an obstacle. On the vine pruning robot d_0 was set to 0.05, margin_rate was 0.25 and $m_{\max}$ was 0.05.

Table 4.4: How the specialised collision detector identifies that pairs of objects are not intersecting when the input robot state is not in collision.

	Mean [%]
Sweep and prune	91
Head bounding volume	6.2
Capsule bounding volumes	2.1
Pairwise checks performed	0.23

randomness in the cut point positioning and path planning algorithms.

Vineyard trials Vineyard tests were performed on one row of vines at Lincoln University. Footage for the robot attempting to prune one plant is available at `http://hilandtom.com/vines.mov`. The robot makes contact with each of the six vines that it is meant to prune during a swipe motion, however, only two of the vines are cut while the remaining four get brushed aside. The failed cuts are due to a combination of the accumulated errors in the sensing of the vine and the positional errors in actuation of the UR5 robot arm.

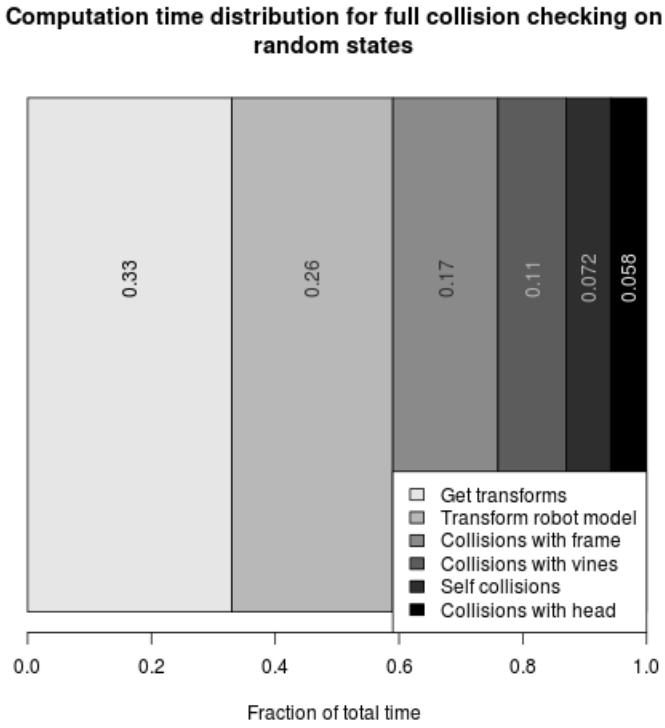
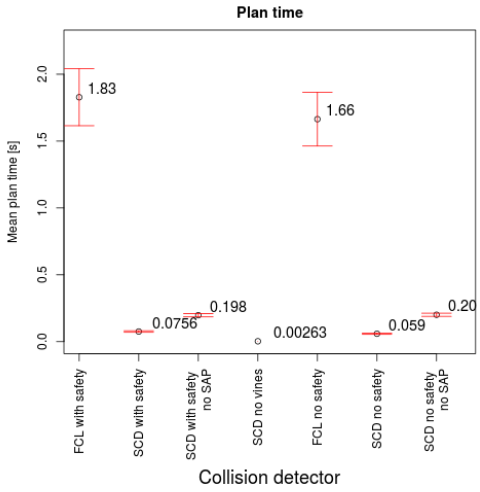
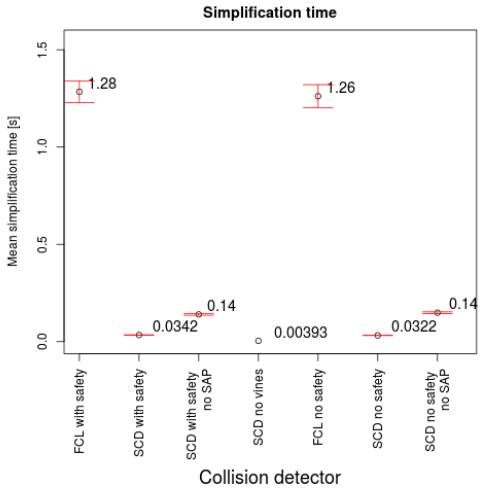


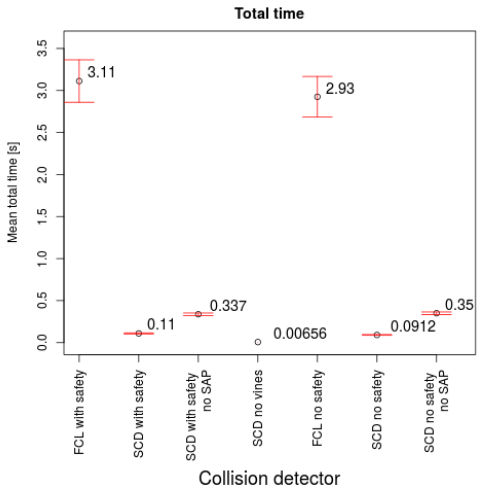
Figure 4.8: Breakdown of computation time for specialised collision detector.



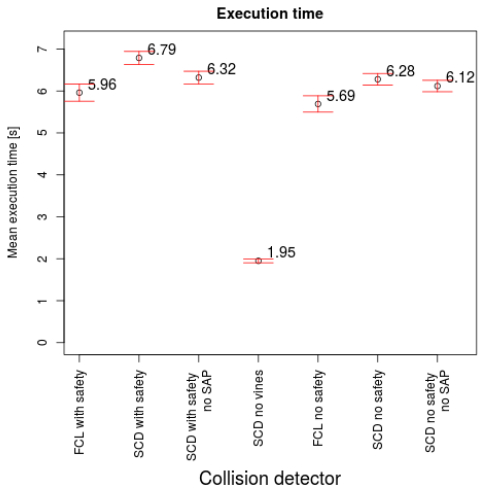
(a) Time to compute a collision free path with RRTConnect.



(b) Time to shortcut and smooth a collision free path.



(c) Total computation time to find a collision free path and simplify it.



(d) Time required for the UR5 robot arm to execute the simplified path.

Figure 4.9: Computation and execution times when using FCL or the proposed specialised collision detector (SCD). The specialised collision detector was tested with and without the sweep and prune (SAP) heuristic enabled. Error bars show the 95% confidence interval.

4.5 DISCUSSION

The specialised collision detector is fast because it performs few pairwise collision checks, as shown in Tab. 4.4. This is because the sweep and prune algorithm allows most pairwise object collision checks to be bypassed. Sweep and prune works well because most of the parts of the robot arm are closer to the robot's base than almost all of the obstacles for most robot configurations.

Putting a sphere bounding volume around the vines head region worked well because it was a tight fit and the head region had a large number of spheres. Without this bounding volume all spheres that were not removed by sweep and prune would need to be checked.

Each capsule was bounded with a sphere because sphere intersection tests are cheaper than capsule intersection tests. This saved less pairwise checks than the bounding sphere around the head region because it only bounded one object. The collision detector could be further improved by bounding multiple capsules with one volume, e.g. by bounding an entire cane with one capsule, but we did not do this because most of the computation time was spent elsewhere computing link transforms (Fig. 4.8).

Most of the collision detection time is spent getting the robot link transforms for the input state and applying these transforms to the robot collision model as shown in Fig. 4.8. This is because very few pairwise intersection tests are performed because of sweep and prune, as shown in Tab. 4.4. Performing intersection tests between the robot arm and robot frame, which only has six planes, takes about the same amount of time as testing the arm against the entire vine. This is because sweep and prune is not applied to these intersection tests.

Using the specialised collision detector in path planning provides a 28 times speed-up compared to using FCL without increasing the robot's execution time (Fig. 4.9). Planning times with the specialised collision detector are also 12 times faster than those by Lee et al. [Lee et al 2014] who report a mean time of 1.5 seconds for a success rate of 80% for their precision weed-spraying task with a UR5 robot arm.

Using adaptive safety margins allowed the use of a large safety margin on some parts of the vine as shown in Fig. 4.3. This provides a guaranteed minimum clearance between the robot arm and parts of the vine away from cuts in computed paths. Using the safety margin caused a small increase in path planning times because the vine became larger. The safety margin relies on having an accurate model of the vines around cuts. The real robot still collides with vines when the 3D model is inaccurate near cuts because the adaptive safety margin is small in those places.

The specialised collision detector exploits the properties of grape vines for the task of cane pruning that allows planning to be performed very quickly, despite the complexity of the environment. Sweep and prune broad-phase step was used to exploit the fact that vines were always positioned far from the robot's base to reduce the required number of intersection tests. Spherical bounding volumes that were used to encapsulate canes and the head region further reduced the number of intersection tests required. Modelling the canes as capsules meant that the remaining intersection tests that had to be performed were fast. Other applications domains also have properties that can be used to optimise planners in a similar way

4.6 SUMMARY

Efficient motion planning is important for robot arms so they can work productively. Standard methods for motion planning are often slow in complex environments because the collision detectors they use are inefficient. By customising a collision detector for the environment a 28 times speed-up in path planning times was obtained. This specialised collision detector performed well on grape vines because it performed very few pairwise intersection tests. The pairwise tests it does perform are fast because the grape vines and robot arm are modelled with capsules and spheres which provide fast intersection tests. Using this fast collision detector approach means that shorter paths can be computed using optimizing planners in the following chapters.

Chapter 5

A COMPARISON OF SAMPLING BASED PLANNERS FOR A VINE PRUNING ROBOT ARM

5.1 INTRODUCTION

Sampling based path planners are the most widely used for robot arm path planning. These path planners use different heuristics and/or data structures to efficiently find collision free paths. Selecting the appropriate path planner for a particular task is important because it influences how efficiently the robot can operate.

Different path planners often perform well for certain classes of problems, and it is still unknown which types of path planner are well suited to each of these problem classes [Moll and Sucan 2015]. This means that it is often difficult to predict how a particular path planner will perform on a specific problem without testing it. In this chapter 17 commonly used path planners are compared to determine which ones would have the best performance on the vine pruning robot.

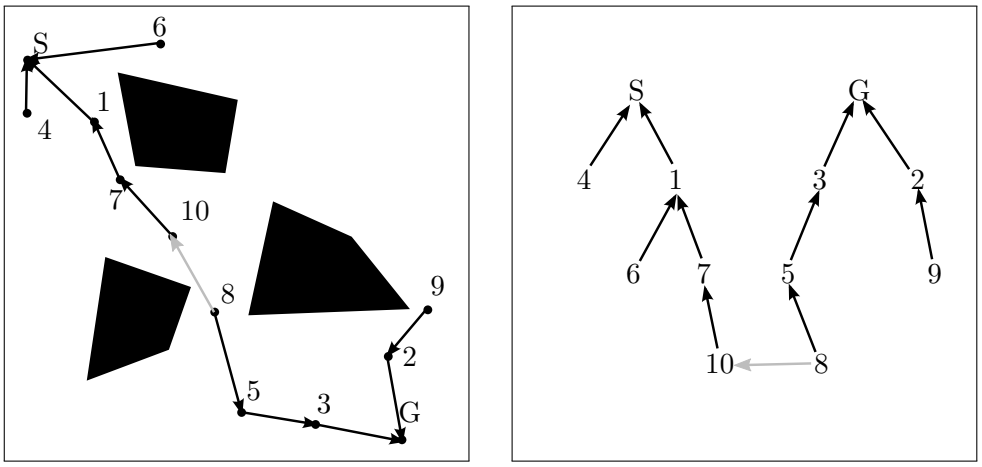
5.2 SAMPLING BASED PATH PLANNERS TESTED

A number of different sampling based path planners exist. These planners are often separated by their sampling strategy, search directionality (e.g. bidirectional) or use of lazy collision evaluation. In many cases a planner with a specific sampling strategy will have bidirectional and/or lazy variations.

Some path planners guide their search toward less-explored regions of configuration space. The vertices in their tree can be projected on to a lower dimensional grid to

estimate their coverage of configuration space, and determine which areas of configuration space are less-explored. The projection can be specified by the user, but random projections have been shown to work well in practice [Sucan and Kavraki 2009b].

Bidirectional planners search from the start to the goal, and from the goal to the start as shown in Fig. 5.1. This is common in single-query planners, where one tree will be rooted at the start vertex and another at the goal vertex. Many bidirectional planners use a *connect* heuristic [Kuffner and Lavalle 2000], where straight line connections are attempted between the two trees when a new vertex is added to either tree. This heuristic is suited to problems where there are large regions of free (configuration) space between the start and goal trees.

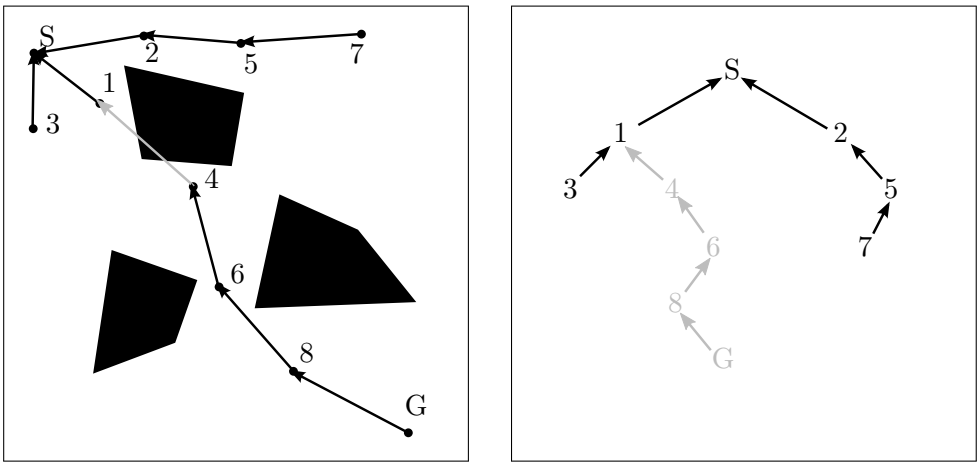


(a) Path plan from a bidirectional search. Connecting motion is shown in light gray.

(b) Start (S) and goal (G) trees of motions with the connecting motion in light gray.

Figure 5.1: Path plan and corresponding tree of motions for a bidirectional search. Nodes represent points in configuration space and lines represent straight-line motions in configuration space. Numbers indicate the order that nodes were sampled.

Path planners can spend a large amount of time collision checking path segments that are not contained in the final solution. Lazy path planners attempt to reduce this by delaying collision checking until a path has been found between the start and goal. If this path is found to be collision-free the path planner terminates. If one or more segments are found to be in collision, the planner prunes child vertices and continues searching for a new path as shown in Fig. 5.2.



(a) Lazy path plan with an invalid path between milestones one and four.

(b) Tree of motions corresponding to lazy path plan. Motions and milestones in light gray are to be removed due to the invalid motion between milestones one and four.

Figure 5.2: Lazy path plan with an invalid motion.

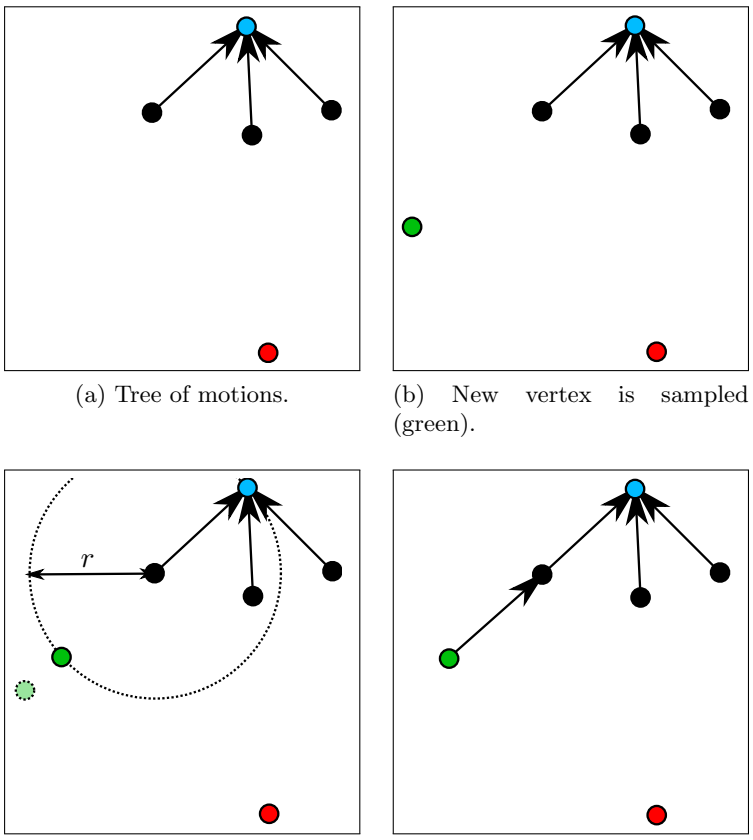
5.2.1 Feasible path planners tested

The RRT is one of the most widely known sampling based path planning algorithms. This planner attempts to find a collision free path by incrementally searching the collision-free part of the robot's configuration space. RRT samples configurations randomly and grows its tree that is rooted at the start vertex as shown in Fig. 5.3. Lazy (LazyRRT) and bidirectional (RRTConnect) variations exist.

The Transition based RRT (TRRT) [Jaillet et al 2010] is a variation of the RRT that is designed to find low cost paths on configuration-space cost maps. TRRT performs transition tests, similar to those used in Simulated Annealing [Kirkpatrick et al 1983], to follow low cost configuration space valleys but remain probabilistically complete. A bidirectional variation, BiTRRT, that uses the connect heuristic exists.

The Expansive Space Trees (EST) [Hsu et al 1999] planner is similar to RRT and was published at a similar time. Unlike RRT, EST selects a vertex to extend its graph from before a new configuration is sampled. This vertex is chosen so that EST is biased toward exploring less-explored regions of configuration space. Bidirectional (BiEST) and Lazy Bidirectional (SBL[1] [Sánchez and Latombe 2003b]) variations exist.

[1]Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking.



(a) Tree of motions.

(b) New vertex is sampled (green).

(c) Vertex is saturated.

(d) Vertex is added to tree.

Figure 5.3: Expansion process for RRT with range r. The start vertex is shown in blue, the goal vertex is shown in red. The green vertex is sampled, saturated, and added to the tree.

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE) [Sucan and Kavraki 2009a] is a single-query planner that uses a discrete grid to guide its search. Configurations in KPIECE's tree are projected onto a two dimensional grid. The search is biased toward cells that are on the boundary of this grid. Bidirectional (BKPIECE) and Lazy Bidirectional (LBKPIECE) variations exist.

The Search Tree with Resolution Independent Density Estimation (STRIDE) [Gipson et al 2013] path planner attempts to expand into regions of configuration space where it has fewer samples, without projecting explored vertices down to a two dimensional grid. New configurations are sampled near existing vertices in its tree where the vertex density is low. The local vertex density is computed using a Geometric Near-neighbour Access Tree (GNAT) [Brin 1995].

Path Directed Subdivision Trees (PDST) [Ladd and Kavraki 2005] are a single query planner that attempts to explore less explored regions of configuration space using a projection. Unlike most other sampling based planners, PDST expands from a randomly selected point on an edge in its graph, rather than expanding from a vertex.

5.2.2 Asymptotically optimal path planners tested

RRT* [Karaman and Frazzoli 2011] is the asymptotically optimal variation of RRT. It continues to optimise solutions until a time limit is reached. RRT* explores configuration space in a similar way to RRT, except new vertices are added into RRT*'s tree. RRTConnect* [Akgun and Stilman 2011, Klemm et al 2015, Jordan and Perez 2013] is the bidirectional variation of RRT*.

Fast Marching Trees (FMT) [Starek et al 2015] is a batch-sampling planner. In the original implementation all of the sampling was performed at the start of planning. FMT then performs a recursion to find a low-cost collision free path from the start to the goal. Collision checking is performed during this recursion. The experiments in this chapter use a version of FMT that allows new samples to be drawn if no collision-free path was found using the initial samples. FMT was designed to perform well in high-

dimensions, and its time complexity scales better than RRT*'s with the configuration space's dimension.

Bidirectional Fast Marching Trees (BFMT) [Starek et al 2015] are a bidirectional variation of FMT. Like FMT*, batch sampling is performed at the start of planning. Two recursions are then performed, one from the start one from the goal, to find a collision free path from the start to the goal. Unlike many other bidirectional sampling-based planners, BFMT does not use a connect heuristic to attempt to join the start and goal trees. Instead, these trees are joined when the two trees both share a vertex that they both added during their recursion.

5.3 EXPERIMENT AND RESULTS

The path planners in Tab. 5.1 were tested on the vine pruning robot. The parameters are shown in Tab. 5.2. The range parameter for each planner was determined by running the planners for one second on 56 cuts and choosing the range value that gave the best success rates, in the event of a tie the value resulting in the lower average computation time was selected.

Table 5.1: Path planners tested

Name	**Uses a projection**	**Bidirectional**	**Lazy**	**Uses connect**
RRT	No	No	No	No
RRT*	No	No	No	No
RRTConnect	No	Yes	No	Yes
RRTConnect*	No	Yes	No	Yes
LazyRRT	No	No	Yes	No
TRRT	No	No	No	No
BiTRRT	No	Yes	No	Yes
EST	No	No	No	No
BiEST	No	Yes	No	Yes
SBL	No	Yes	Yes	Yes
KPIECE	Yes	No	No	No
BKPIECE	Yes	Yes	No	Yes
LBKPIECE	Yes	Yes	Yes	Yes
STRIDE	Yes	No	No	No
PDST	Yes	No	No	No
FMT	No	No	No	No
BFMT	No	Yes	No	No

Table 5.2: Parameters used for each planner in experiments. Each cell contains the parameter name and its value to achieve a compact layout.

Planner	**Parameters**							
RRT	range: 1.5	goal bias: 0.05						
RRTConnect	range: 0.5							
LazyRRT	range: 3.0	goal bias: 0.05						
TRRT	range: 3.0	goal bias: 0.05	temp change factor: 0.1	frontier threshold: 19689	frontier node ratio: 0.1			
BiTRRT	range: 1.0	temp change factor: 0.1	frontier threshold: 19689	frontier node ratio: 0.1				
EST	range: 2.5	goal bias: 0.05						
BiEST	Range: 1.5							
KPIECE	range: 0.5	goal bias: 0.05	border fraction: 0.9	failed expansion cell score factor: 0.5	min valid path fraction: 0.2			
BKPIECE	range: 1.0	border fraction: 0.9	failed expansion cell score factor: 0.5	min valid path fraction: 0.2				
LBKPIECE	range: 1.0	goal bias: 0.05	border fraction: 0.9	failed expansion cell score factor: 0.5	min valid path fraction: 0.2			
PDST	goal bias: 0.05							
STRIDE	range: 1.0	goal bias: 0.05	use projected distance: false	Degree: 16	min degree: 12	max degree: 18	max points per leaf: 6	estimated dimension: 6
SBL	range: 1.5							
FMT	num samples: 10000	extended FMT: true	use heuristics: true	rewire factor: 1.1	cache cc	true	nearest K: true	
BFMT	num samples: 10000	extended FMT: true	use heuristics: true	rewire factor: 1.1	cache cc	true	nearest K: true	
RRT*	range: 1.5	goal bias: 0.05	rewire factor: 1.1	focus search: true	prune threshold: 0.05			
RRTConnect*	range: 1.5	rewire factor: 1.1	focus search: true	prune threshold: 0.05				

The path planners were then tested in simulation on 312 cuts each for a maximum of one second using data collected from Sauvignon Blanc vines at Lincoln University. For each plant the robot arm was started at joint positions [0,-20,0,0,0,0]. Path plans were then computed between successive cut-points. If the planner failed to compute a plan the failure was recorded and the start position of the robot arm was not changed. The success rates for each path planner, that is how often they were able to find a collision free path within the one second timeout, are shown in Fig. 5.4.

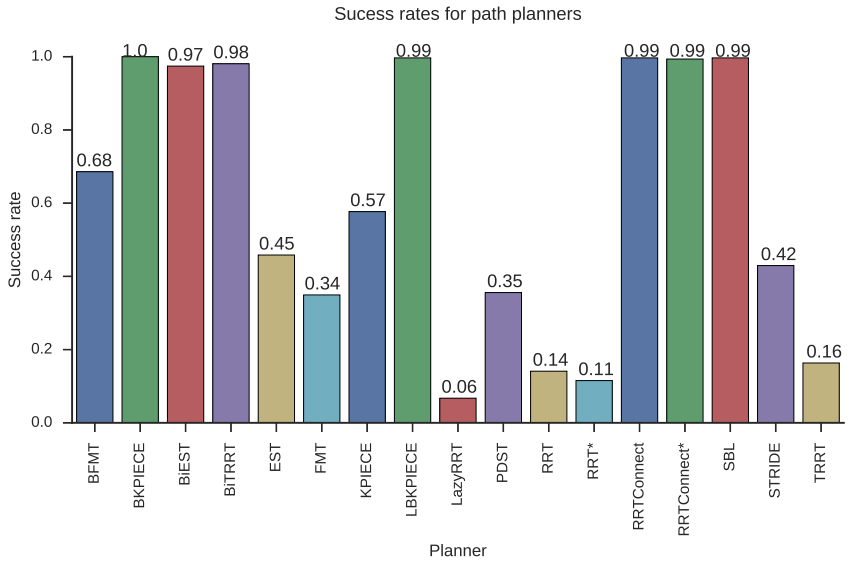


Figure 5.4: Path planner success rates for vine pruning experiment.

Fig. 5.5 shows the computation times for each of the planners. RRT* and RRTConnect* always report a computation time near the one second timeout. This is because they use all of the allowed planning time to optimise solutions. The other planners terminated as soon as a solution was found.

Fig. 5.6 shows the Euclidean length of the paths found by the planners. No length is reported for queries where a solution was not found. This means that it is difficult to compare two planners that have low success rates, because they might be reporting lengths for paths found on different queries.

Fig. 5.7 shows the execution times for the paths found by each of the planners. Execution times are only reported for queries where the planner was able to find a

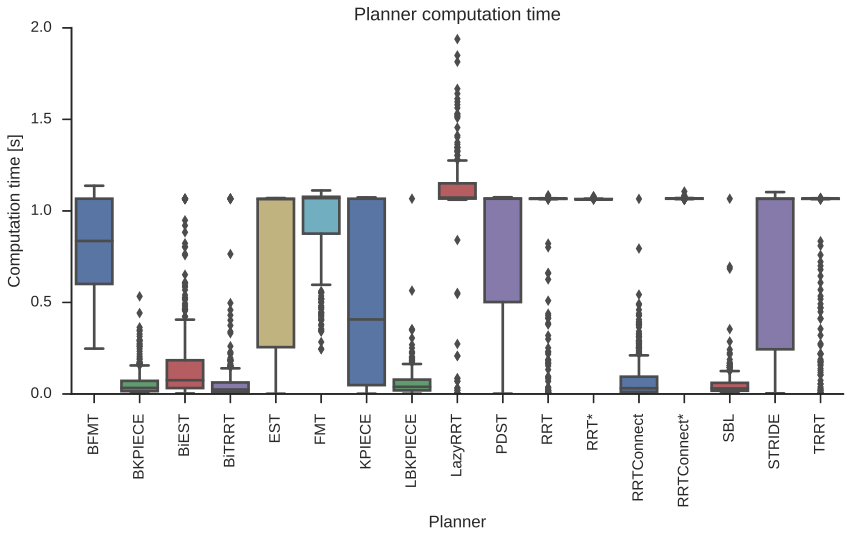


Figure 5.5: Path planner computation times for vine pruning experiment. Some planners took more than the one second timeout because they had to perform additional clean up operations. Whiskers extend to 1.5 times the interquartile range from the upper/lower quartiles.

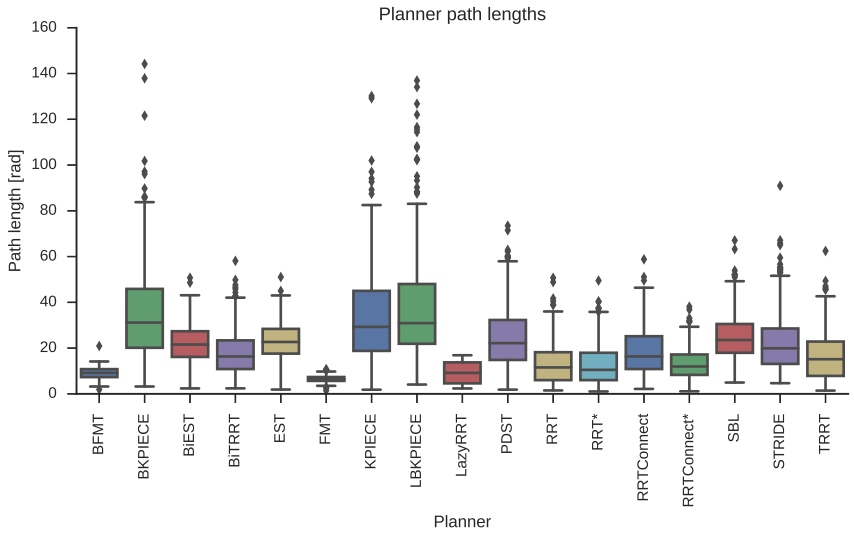


Figure 5.6: Length of paths found by planners in successful queries for the vine pruning experiment. Whiskers extend to 1.5 times the interquartile range from the upper/lower quartiles.

collision free path. The execution time is how long it would take the UR5 vine pruning robot arm to follow the found path and does not include the time it would take to perform the short swiping cut motion. This is important because it directly effects the efficiency of the pruning robot, because a significant portion of the pruning time is spent waiting for the robot arm to follow paths.

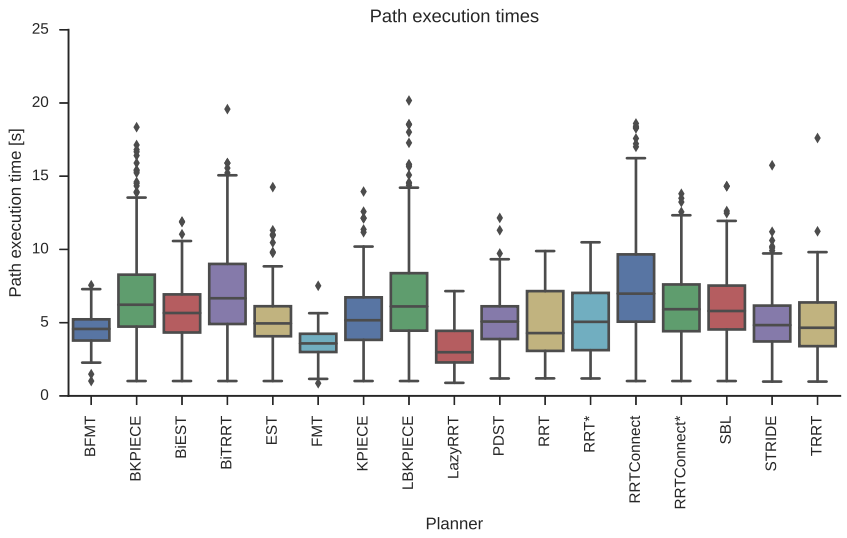


Figure 5.7: Execution times of paths found by planners in the vine pruning experiment. Whiskers extend to 1.5 times the interquartile range from the upper/lower quartiles.

5.4 DISCUSSION

Bidirectional planners that use a connect heuristic (BKPIECE, BiEST, BiTRRT, LBKPIECE, RRTConnect, RRTConnect*, SBL) all had success rates above 95% as shown in Fig. 5.4. In many cases the single-directional variations of these planners had poor success rates, e.g. RRT had a success rate of 14% but RRTConnect had a success rate of 99%. BFMT, the only bidirectional planner that did not use a connect heuristic, had a success rate of 68% which is significantly lower than the success rates of the other bidirectional planners that all used the connect heuristic.

The path planners using the connect heuristic, other than RRTConnect*, all had low computation times as shown in Fig. 5.5. This means that they were consistently able to quickly find collision free paths. Path planners with low success rates had high computation times, because they often reach the one second timeout. LazyRRT often exceeded the one second timeout because it performed a lot of clean-up operations before termination.

RRTConnect* managed to find slightly shorter paths on average than the other planners using the connect heuristic with high success rates (BKPIECE, BiEST, Bi-

TRRT, LBKPIECE, RRTConnect, SBL) as shown in Fig. 5.6. Path lengths were only reported for queries where the planner successfully found a collision free path. This means that the reported lengths for planners with low success rates, e.g. LazyRRT and FMT, may misrepresent the actual performance of the planner. This means that only planners with high success rates should be compared for the length of path that they found.

The execution times were reasonably similar between planners as shown in Fig. 5.7. FMT, BFMT and LazyRRT reported slightly lower execution times, but these planners had low success rates so may have mainly been finding solutions to queries where the other planners were also finding fast to execute paths. Most path planners that had high success rates report median execution times near five seconds. Interestingly, SBL reported similar path execution times to RRTConnect* despite reporting higher path lengths and not spending additional time optimising solutions (SBL terminates as soon as a solution is found, while RRTConnect* uses all of the available planning time to optimise solutions).

The execution times reported by the planners with high success rates were significantly higher than their reported computation times. This means that if they were used on the vine pruning robot most of the time would be spent waiting for the robot arm to follow paths, rather than waiting for planners to compute collision free paths. These execution times need to be reduced to make the vine pruning robot operate more efficiently. They could be reduced by using planners that are designed to find short paths.

One approach to finding shorter paths could be to allow RRTConnect* to run with longer computation times. Another could be to improve the convergence rate of RRTConnect*, so that it finds shorter paths for a given computation time. These shorter paths may be faster to execute, meaning that the vine pruning robot could operate more efficiently.

Bidirectional path planners that made use of the connect heuristic were found to have higher success rates and lower computation times when compared to the other

planners. Bidirectional planners with the connect heuristic are well suited to the problem of pruning vines with a UR5 robot arm because the planning queries often start and end in cluttered areas (cut points where there are lots of vines to avoid), but there are often large collision-free regions of configuration space in between the start and end (where the robot arm pulls back from the vine). These planners are able to find paths that navigate out of the cluttered start and end positions by incrementally growing trees, and the connect heuristic quickly finds collision free path segments over the large collision-free regions separating the start and end regions.

5.5 SUMMARY

A number of commonly used sampling-based path planners were tested on a vine pruning robot arm. Bidirectional planners that used the connect heuristic were found to be the only planners that consistently found collision free paths within the one second of allowed planning time. Path planners that did not use the connect heuristic, including the bidirectional BFMT planner, were found to have poor success rates for the vine pruning task.

Planners using the connect heuristic were all found to have sub-second computation times, while they found paths that took substantially more time to execute. This means that the vine pruning robot would spend significantly more time executing paths than computing them. The next step to improving the efficiency of the vine pruning robot is to develop a path planner that is capable of finding shorter paths that are fast to execute.

One approach to finding short paths that are fast to execute is to use an asymptotically optimal planner. The results of the experiments in this chapter suggest that an asymptotically optimal planner that uses the connect heuristic would be well suited to the vine pruning problem, although one such planner (RRTConnect*) was tested and found paths with reasonably large execution times (although it did outperform the other asymptotically optimal planners tested and was the only AO planner to consistently find solutions). In the next chapter I develop an approach to speeding up

RRTConnect*, so that it can find shorter paths that are faster to execute given the same amount of computation time.

Chapter 6

INTEGRATING A LOCAL OPTIMISER WITH ASYMPTOTICALLY OPTIMAL PATH PLANNER

6.1 INTRODUCTION

Many sampling based path planning algorithms, e.g. RRT and PRM, can find feasible paths quickly but are not guaranteed to find the shortest path, regardless of time available. Recent work on optimal planning, e.g. RRT*[Karaman and Frazzoli 2011] and PRM* [Karaman and Frazzoli 2011], extend these algorithms to guarantee asymptotic optimality, however, these algorithms may require a long time to find a good path. This chapter considers speeding-up existing optimal planning algorithms for practical applications where the computation time available for planning is limited.

This chapter proposes integrating a local 'short-cutting' optimiser with the RRT-Connect* [Akgun and Stilman 2011, Jordan and Perez 2013, Klemm et al 2015] (a bidirectional variation of RRT*) asymptotically optimal path planner to quickly improve intermediate solutions. The purpose of this planner is to find paths with short cycle times (computation time + execution time). This approach is evaluated on the vine pruning robot arm and cubicle picking tasks.

6.2 OPTIMAL PLANNING BACKGROUND

Path planners can be categorised as feasible planners [Hsu et al 1999, Kavraki et al 1996, LaValle 1998, LaValle 2001], or optimizing planners [Gammell et al 2015, Janson et al 2015, Karaman and Frazzoli 2011, Salzman and Halperin 2016]. Feasible planners

attempt to quickly find a solution and terminate as soon as one is found. Feasible planners can return poor, e.g. long, solutions because they do not perform optimization. Optimizing planners attempt to find high-quality, e.g. short, solutions within a set computation time or number of iterations. Some of these optimizing planners are asymptotically optimal and will converge to the optimal solution eventually [Gammell et al 2015, Janson et al 2015, Karaman and Frazzoli 2011]. Other optimizing planners are asymptotically near-optimal and will converge to a near-optimal solution eventually [Dobson and Bekris 2014, Salzman and Halperin 2016]. In this chapter an approach is proposed for speeding up the convergence of optimizing planners.

A key requirement for many popular asymptotically optimal path planners, e.g. Batch Informed Trees (BIT*) [Gammell et al 2015], RRT*, PRM*, is that when a new vertex v is inserted into the planner's graph G, edges are formed between v and vertices within its neighbourhood V_{nbh}. V_{nbh} can be defined as all vertices in G within a radius r_{nbh} of v, or as the k_{nbh} neighbours of v. Minimum values for r_{nbh} and k_{nbh} depend on the number of vertices in G [Karaman and Frazzoli 2011]. Some multiple query planners, e.g. PRM*, form edges between v and all its neighbours where a collision-free path exists. Single query planners perform a *rewiring* step that joins v to a sub-set of it's neighbours without adding cycles their graph.

Fig. 6.1 illustrates how RRT* rewires a new vertex v into its graph G (Fig. 6.1a). The new vertex is initially connected to its nearest neighbour in G (Fig. 6.1b). The neighbourhood of v in G is computed, v is then connected to a different vertex to minimise the cost to arrive from the start vertex (Fig. 6.1c). The edges of neighbouring vertices are changed if the path through v provides smaller cost to arrive at that particular neighbour (Fig. 6.1d). This rewiring approach is guaranteed to not introduce cycles into G [Karaman and Frazzoli 2011].

As optimising planners converge to the optimal solution they can spend more time processing samples that cannot possibly be used to improve on the best solution [Gammell et al 2014]. This can be remedied by focussing the planner's search to useful regions of configuration space [Gammell et al 2014] and rejecting new samples that cannot be

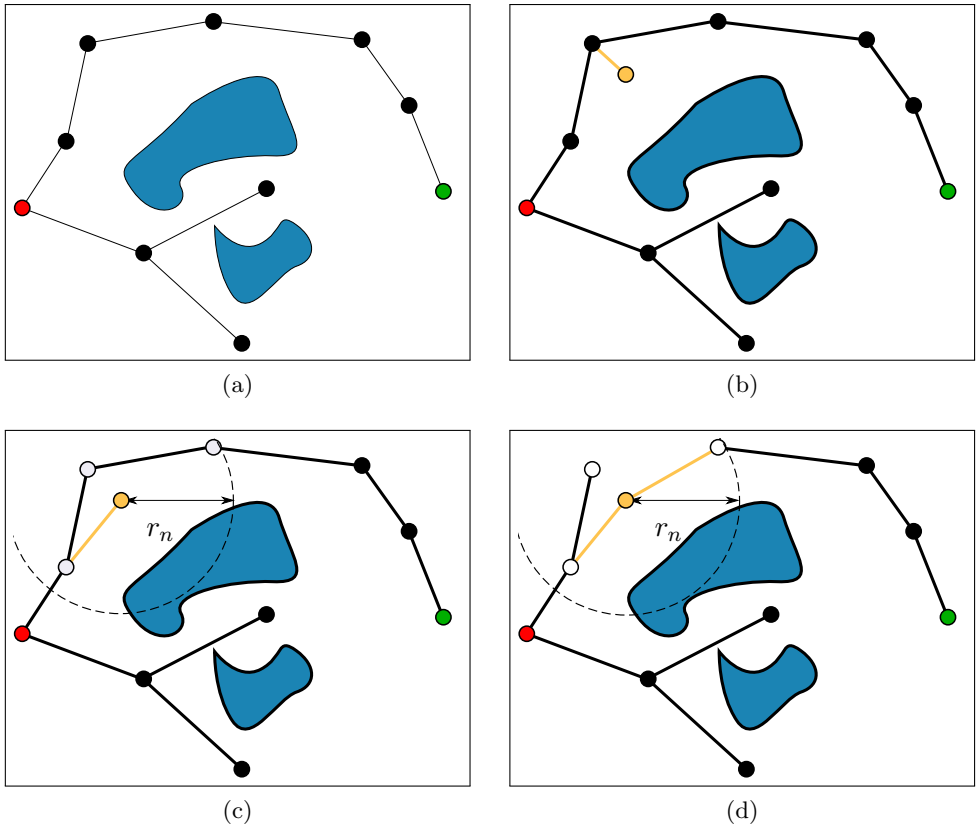


Figure 6.1: RRT* insertion of the yellow vertex. The start vertex is shown in red and the goal vertex is shown in green. r_n denotes the neighbourhood radius and vertices within this neighbourhood are shown with a white fill.

used to improve on the planner's best solution [Akgun and Stilman 2011] without sacrificing the optimality properties of the planner. In this chapter RRTConnect* is tested with these heuristics enabled.

Local optimization algorithms can be used with path planners to quickly improve path quality, e.g. reduce length. These algorithms often rely on a path planner to provide an initial solution. This initial solution influences the quality of the optimised path because these algorithms only optimize locally. Short-cutting [Berchtold and Glavina 1994, Geraerts et al 2007, Hauser and Ng thow hing 2010] for reducing path length and sequential convex optimization approaches [Kalakrishnan et al 2011, Schulman et al 2014, Zucker et al 2013] have been shown to work well on robot arms.

A common approach to finding short paths is to find an initial collision-free solution with a feasible planner, e.g. RRTConnect [Kuffner and Lavalle 2000] (a bidirectional RRT), and to optimise this path with a local optimiser e.g. short-cutting. Another approach is to perform multiple restarts of the feasible planner, optimise each solution and return the best solution as shown in Fig. 6.2. This has been shown to work well in empirical experiments when compared to asymptotically optimal planners [Luo and Hauser 2014]. The approach of RRTConnect* integrated with short-cutting is compared to multiple restarts of RRTConnect with short-cutting, as well as one run of RRTConnect with short-cutting.

There has been some recent interest in combining asymptotically optimal path planners with local optimisers to speed up convergence to optimal solutions. Choudhury et al [2016b] use the CHOMP local optimiser to avoid collisions in edges between vertices in their Regionally Accelerated Batch Informed Trees (RABIT*) planner. This differs from the approach detailed in this chapter because the approach in this chapter uses a local optimiser to improve a complete path, rather than individual segments.

A recent preprint proposed 'interleaving' the use of a global asymptotically optimal path planner with a local optimiser [Kuntz et al 2016]. The global planner is used to explore the robot's configuration space and the local optimiser is used to quickly improve solutions. The local optimiser is invoked every time the global planner finds

```
1: function MRRTConnect+S(v_start, V_goal, termination_condition)
2:     L_best ← ∞
3:     p_best ← NULL
4:     do
5:         p ← RRTConnect(v_start, V_goal)
6:         p ← Shortcut(p)
7:         L ← The length of p
8:         if L < L_best then
9:             L_best ← L
10:            p_best ← p
11:        end if
12:    while not termination_condition
13:    return p_best
14: end function
```

Figure 6.2: Multiple restarts of RRTConnect with short-cutting.

a better solution. The optimised path is then placed into the planner's graph without forming edges to existing vertices within the graph i.e. no rewiring step is performed. This means that the interleaving approach is only asymptotically optimal for some planners, e.g. PRM*, and special consideration must be given to only include vertices added by the global planner when calculating the neighbourhood size.

This chapter builds on the interleaving approach in two ways: Firstly, locally optimised paths are rewired back into the planner's graph to preserve the asymptotic optimality of the global planner. Secondly, the local optimiser is only invoked when the global planner has substantially improved on the last locally optimised path. This prevents the local optimiser being invoked every time the global planner has made a small incremental improvement to the last optimised path.

6.3 INTEGRATING RRTCONNECT* WITH A SHORT-CUTTING LOCAL OPTIMISER

Our approach speeds up RRTConnect* by integrating a short-cutting local optimiser. Good intermediate solutions found by RRTConnect* are shortcut and inserted into RRTConnect*'s graph as shown in Fig. 6.3. v_{start} and V_{goal} represent the start vertex and

goal vertices for the planning query. Planning continues until the termination_condition expires, e.g. this could be an iteration count or a timeout.

To maintain asymptotic optimality, vertices from the short-cut path are rewired into RRTConnect*'s graph. To ensure that the short-cut path is recoverable through RRTConnect*'s graph, the neighbourhood of each of the path's vertices is expanded to include the path's previous vertex as shown in Fig. 6.4. After path insertion the length of the best path through the planner's graph L'_{best} is:

$$L'_{\text{best}} \leq \min(L_{\text{best}}, L_{\text{path}}) \tag{6.1}$$

Where L_{path} is the length of the path that was inserted and L_{best} is the length of the shortest path before the new path was inserted. The Shortcut routine is terminated after a fixed number of iterations in this chapter.

The approach in this chapter can be extended to other planners by changing the planner used in Fig. 6.3 (line 4). The InsertPath method may have to be altered for use with planners such as PRM* that do not perform rewiring.

6.4 EXPERIMENTS

To test the proposed approach, the performances of the planners in Tab. 6.1 is compared to RRTConnect* integrated with a short-cutting local optimiser using vine data collected from Lincoln University. These planners are tested on the vine pruning and cubicle picking robots detailed in Chapter 3.

In both trials, RRTConnect* with and without short-cutting was configured to minimise Euclidean path length. Path planner parameters are shown in Tab. 6.2 for the vine pruning experiments and Tab. 6.3 for the cubicle picking experiments. The number of iterations performed by the short-cut optimiser was calculated as:

$$N_{\text{iterations}} = \text{SCF} \times N_{\text{vertices}} \tag{6.2}$$

```
1: function RRTConnect*+S(v_start, V_goal, opt_threshold, termination_condition)
2:     C_last_opt ← ∞
3:     do
4:         G ← RRTConnect*(v_start, V_goal, G)    // One iteration
5:         best_path ← Best cost path from v_start to V_goal through G
6:         C_best ← Cost of best_path
7:         if (C_last_opt − C_best) / C_last_opt > opt_threshold then
8:             p_optimized ← Shortcut(p_shortest)
9:             G ← InsertPath(G, p_optimised, v_start)
10:            C_last_optimized ← C_shortest
11:        end if
12:    while not termination_condition
13:    return Lowest cost path from v_start to V_goal through G
14: end function
```

Figure 6.3: RRTConnect* with short-cutting. The blue lines show our proposed changes to RRTConnect*.

Table 6.1: Summary of planners evaluated

Name	**Description**
RRTConnect+S	One run of RRTConnect followed by short-cutting optimizer
MRRTConnect+S	Multiple restarts of RRTConnect + short-cut where the best path is kept. A leading contemporary approach [Luo and Hauser 2014]. See Fig. 6.2.
RRTConnect*	Asymptotically optimal RRTConnect. Uses the informed heuristics [Gammell et al 2014].
RRTConnect*+S	Our proposed approach. RRTConnect* with informed heuristics that uses short-cutting local optimizer.

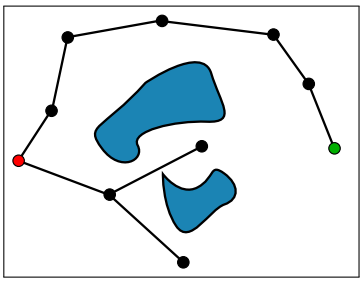
Where SCF is the shortcut count factor parameter and N_{vertices} is the number of vertices in the path. The value for SCF was determined by performing a parameter sweep by planning to a subset of the cuts, as where the rest of the parameters in Tab. 6.2.

Table 6.2: Parameters used in vine pruning robot tests

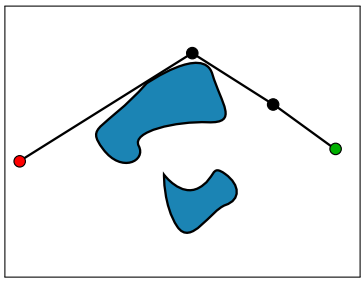
Planner	range	shortcut count factor	local optimization threshold
RRTConnect*	2.5	-	-
RRTConnect* with shortcut	2.5	3.0	0.01
RRTConnect+S	0.5	4.0	-
MRRTConnect+S	0.5	4.0	-

Table 6.3: Parameters used in cubicle picking tests

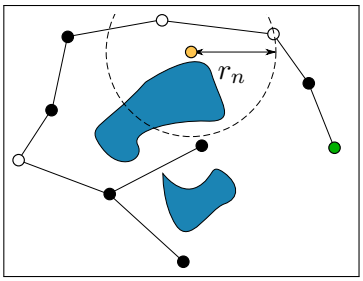
Planner	range	shortcut count factor	local optimization threshold
RRTConnect*	0.5	-	-
RRTConnect* with shortcut	3.0	3.0	0.11
RRTConnect+S	0.5	3.0	-
MRRTConnect+S	0.5	3.0	-



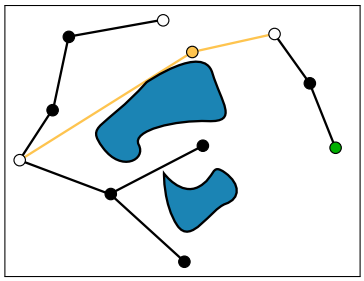
(a) Planner's existing graph G with one solution.



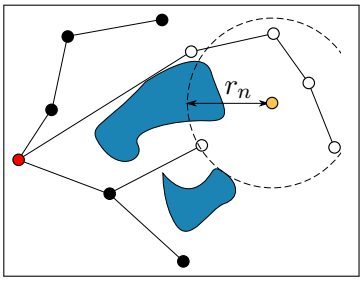
(b) Path to be inserted, p, into G.



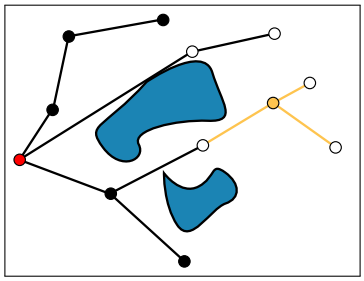
(c) Neighbourhood of vertex 2 of p that has been extended to include previous vertex from path.



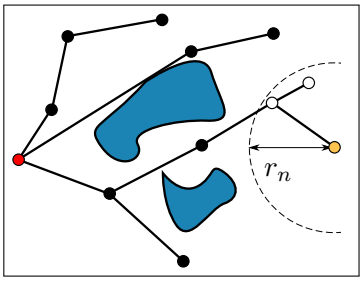
(d) The vertex is added to G and its neighbourhood is rewired.



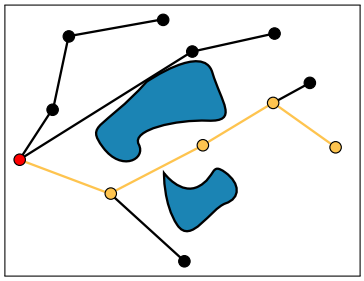
(e) Neighbourhood of vertex 3 from p that has been extended to include previous vertex from path.



(f) The vertex is added and its neighbourhood is rewired.



(g) The final vertex of p is inserted into G, but no edges change.



(h) G after p has been added, the new minimum cost path shown in yellow.

Figure 6.4: Insertion of a path p into a planner's graph G using RRT*'s insertion procedure for the objective of minimising Euclidean path length. The start vertex in G is red and the goal vertex is green. Vertices and the edges that are added/modified are shown in yellow, except for (h) where the final path is shown in yellow. Vertices part of a yellow vertex's neighbourhood have a white fill. r_{n} is the radius that defines the neighbourhood of the yellow vertex.

6.5 RESULTS

For both experiments the Euclidean length (sum of Euclidean lengths of each path segment, in radians), execution time (how long it would take the robot arm to follow the path once accelerations have been assigned in post processing), the number of local optimisations and the cycle time (computation time plus execution time) were recorded. These measurements were taken from the planner in a separate thread as to not interfere with the planner's performance.

Integrating RRTConnect* with a short-cut local optimiser resulted in significant speed-ups as shown in Fig. 6.5 and Fig. 6.6. It resulted in a 24% reduction in cycle time for the vine pruning robot, and a 21% decrease in cycle time for the cubicle picking robot. MRRTConnect+S also performed well in both experiments. These speed-ups result in lower robot cycle times because the computation and execution times are of similar magnitude.

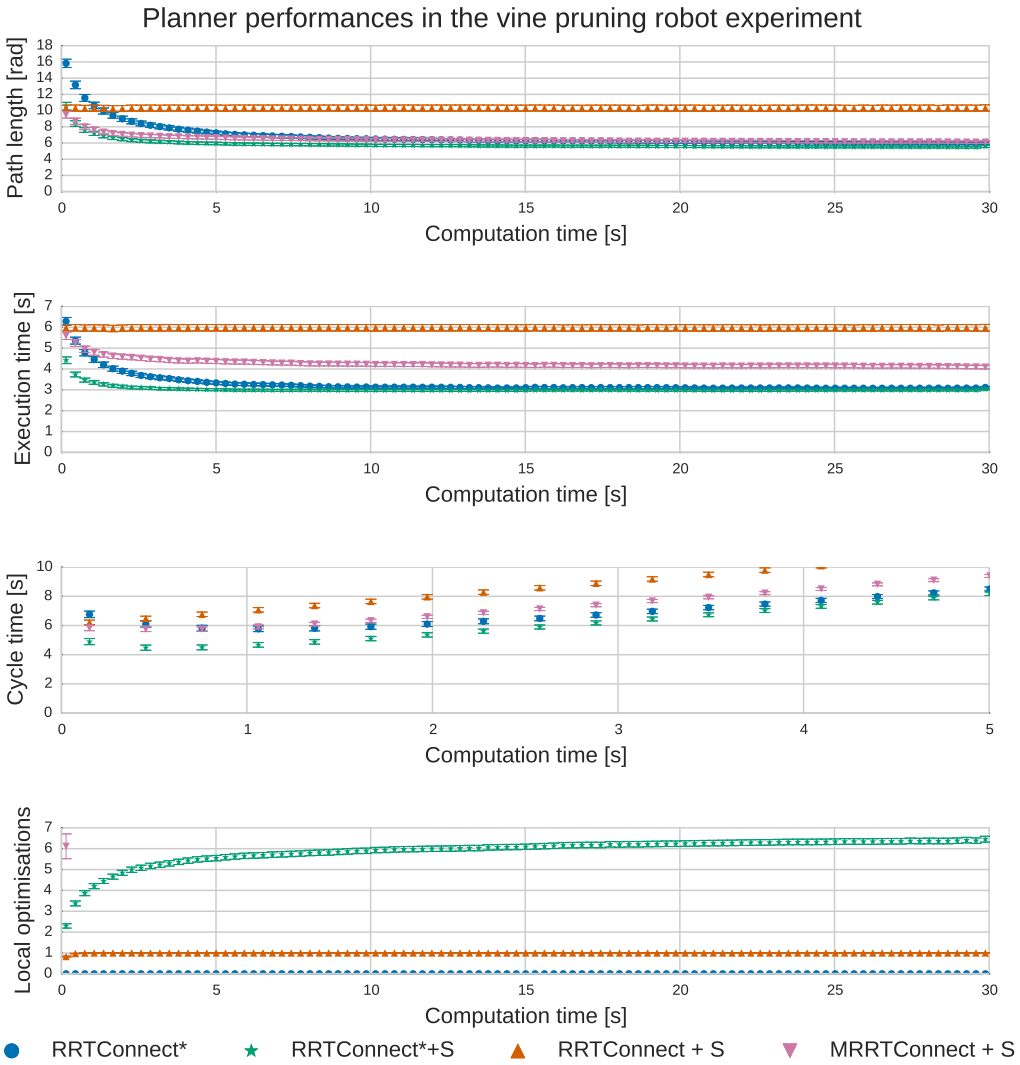


Figure 6.5: Means for 304 successful grape vine planning queries. Error bars show the 95% confidence interval. MRRTConnect+S averaged 905 local optimisations after 30 seconds of planning, it was truncated for clarity. For a fixed time budget RRT-Connect*+S found paths that were faster to execute than RRTConnect*, allowing the robot to have a shorter cycle time.

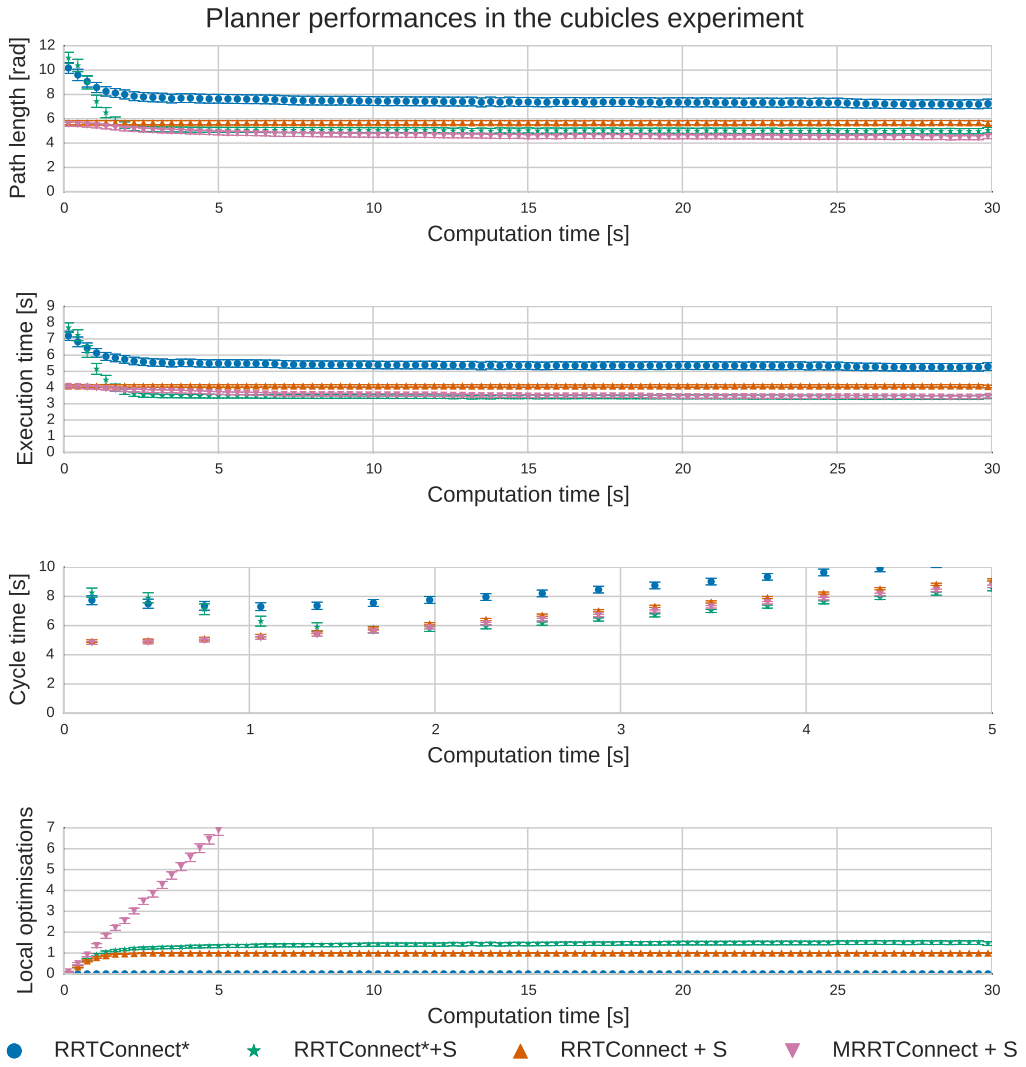


Figure 6.6: Means for 144 cubicles queries. Error bars show the 95% confidence interval. MRRTConnect+S averaged 42.3 local optimisations after 30 seconds of planning, it was truncated for clarity. For a fixed time budget RRTConnect*+S found paths that were faster to execute than RRTConnect*, allowing the robot to have a shorter cycle time.

6.6 DISCUSSION

Integrating RRTConnect* with a short-cut local optimiser resulted in shorter paths being found more quickly compared to not using the short-cut optimiser as show in Fig. 6.5 and Fig. 6.6. This is consistent with the results of a recent preprint [Kuntz et al 2016] where BIT* and PRM* were interleaved with a Lagrangian local optimiser.

In the cubicles experiment RRTConnect*+S only performed around one local optimisation. This is because it tended to find short solutions after one local optimisation and could not improve these solutions enough to invoke the local optimiser again. RRTConnect+S also performed well on this experiment. This suggests that the configuration space for the cubicles experiment is very sparse and optimising a wide range of initial paths could result in a short path.

RRTConnect*+S was sparing with its use of the local optimiser in both experiments as shown in Fig. 6.5 and Fig. 6.6. This is because the local optimiser was only invoked when RRTConnect* has improved the path by a certain threshold (Fig. 6.3). MRRTConnect+S made a lot of calls to the local optimiser because it was called every time a new path was found. It could be expected that the interleaving optimiser [Kuntz et al 2016] to make a lot of calls to the local optimiser because it is invoked every time the global planner (even slightly) improves the path. MRRTConnect+S and the interleaving approach may spend a lot of time in local optimisation if a slow local optimiser is used.

The short-cut optimiser was a good fit for both robots as shown by the good performance of MRRTConnect+S in both experiments. This could be caused by the robots in both experiments robots having sparse configuration spaces. The short-cut optimiser is not a good fit for all problems, especially those where the triangle inequality does not hold. In these spaces it is possible that short-cutting a path may lead to it becoming longer. The path insertion method (see Sec. 6.3) guarantees that the insertion of a poor path does not degrade the quality of any other paths found by the planner.

6.7 SUMMARY

This chapter detailed an approach to speeding up the convergence of the RRTConnect* asymptotically optimal path planner by integrating it with a short-cutting local optimiser. Integrating the local optimiser significantly improved the performance of RRTConnect* for the vine pruning and cubicle picking robot arm tasks. This approach found paths that were 31% faster to execute than paths found by RRTConnect* after three seconds of planning time.

Chapter 7

FINDING SHORTER PATHS FOR ROBOT ARMS USING THEIR REDUNDANCY

7.1 INTRODUCTION

Many robot arms can accomplish one task using many different joint configurations. Often only one of these configurations is used as a goal by the path planner. Ideally the robot's path planner would be able to use the extra configurations to find higher quality paths. In this paper we use the extra goal configurations to find significantly shorter paths that are faster to execute when converted to trajectories compared to a planner that chooses one goal configuration arbitrarily. In a grape vine pruning robot arm experiment our proposed approach reduced execution times by 58%.

Robot arm tasks are specified in their workspace for many applications, e.g. move the end-effector to a specific Cartesian position. These tasks can often be accomplished by the robot arm in many different poses (Fig. 7.1), where a pose is defined as the workspace position of every part of the robot arm (not just the end-effector). Each pose has a different robot configuration. We will define the different configurations that can be used to achieve a task where the robot occupies *different* volumes in the workspace to be the robot arm's *workspace redundancy.*

Some robot arms have joints that are capable of making more than one full revolution, e.g. Universal Robot's (UR) popular UR3, UR5, UR10 and ABB's IRB 2400 robot arms. These robot arms have *configuration space redundancy* where each robot pose can be represented by a number of different configurations that put each of the

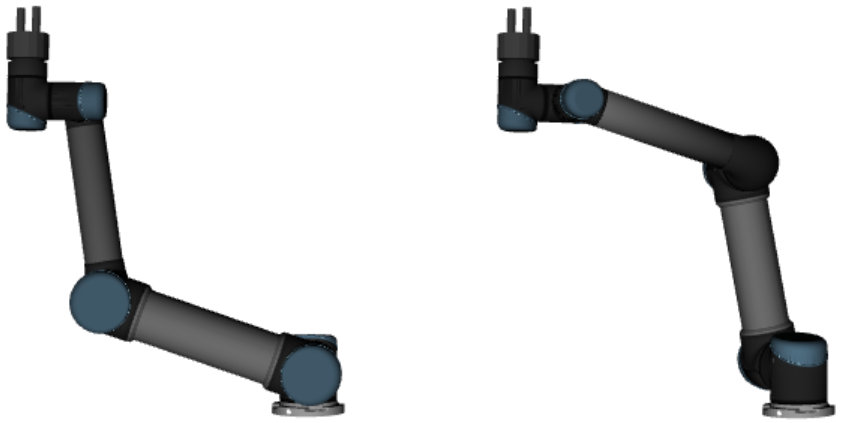


Figure 7.1: Two poses for the UR5 robot arm that put the end-effector at the same position illustrating workspace redundancy. Each of these poses can be achieved with 64 different robot arm configurations where one of more joints are rotated one full revolution, this is the robot arm's configuration space redundancy.

robot's links in exactly the same workspace positions i.e. the robot occupies *exactly* the same volume in the workspace. This means that equivalent configurations cannot be distinguished by looking at the position of the robot's links, the joint positions must be known. For example, each of the UR5 arm poses shown in Fig. 7.1 can be achieved with 64 different configurations (see Sec. 7.3). We will call the configurations that lead to the same robot pose *equivalent configurations*. It follows that some goals can be achieved using a number of different workspace poses of the robot, and each of these workspace poses has 64 equivalent configurations where one or more of the joints are rotated by one revolution. Therefore, the complete set of goal configurations for the UR5 is the product of the number of workspace poses the robot arm can use to achieve the goal and the number of equivalent configurations that it can achieve (which is 64 for the UR5). This chapter investigates the effect of using the robot arm's workspace and configuration space redundancy in path planning to find shorter paths using an asymptotically optimal planner.

7.2 BACKGROUND TO IMPROVING PLANNER PERFORMANCE USING WORKSPACE REDUNDANCY

Path planning is often performed in the robot's configuration space. In many cases the path planner only considers one [Lee et al 2014, Coleman et al 2015, Stilman et al 2007, Hirano et al 2005] of the possibly many configurations that the robot arm can use to achieve its goal.

One approach to exploiting a robot's redundancy is to allow the path planner to use multiple Inverse Kinematic solutions as a goal. This approach has been used with feasible path planners to improve their success rates [Dalibard et al 2009, Ellekilde and Petersen 2013]. This approach is simple, but requires a fixed number of inverse kinematic solutions to be computed before planning begins. Computing a large number of inverse kinematic solutions can be computationally expensive and unnecessary for some planning queries. Using too few inverse kinematic solutions may lower the success rate of the planner.

Inverse kinematic solutions can be instead computed during planning from a *workspace goal region* [Berenson and Ferguson 2009, Berenson et al 2011]. A workspace goal region is a volume of space in the robot's workspace where the robot's end-effector should be placed to satisfy the task requirements. Inverse kinematic solutions satisfying this workspace goal region are computed and added to the planner's goal representation during planning. An advantage of this approach is that more goal configurations are considered on difficult planning queries, when the planner takes a long time to find a solution. Fewer inverse kinematic solutions are computed on simpler queries when the planner quickly finds a path. A summary of the different kinds of goals covered is shown in Tab. 7.1.

Some planners are capable of finding paths to goals in the robot's workspace without using an inverse kinematics routine [Vande Weghe et al 2007]. Exploration is guided by a local controller that uses the transpose of the robot's Jacobian to minimise the distance between explored configurations and the robot's goal specified

Table 7.1: Goal types and descriptions

Goal type	**Description**
Goal configuration	A point in configuration space that is the goal for the planner. The planner should find a path that ends at this point in configuration space.
Goal configurations	A set of configurations. The planner should find a path that ends at one of these.
Workspace goal	A goal that is defined in the robot's workspace e.g. a Cartesian position for the end-effector. The planner should find a path that finishes at a configuration that puts the end-effector in a position that satisfies this goal.
Workspace goal region	One or more volumes in the robot's configuration space that would satisfy a workspace goal for the robot. The planner should find a path that finishes at a configuration within one of these volumes.

in the workspace. A number of other approaches that use problem-specific heuristics [Bertram et al 2006, Drumwright and Ng thow hing 2006, Keselman et al 2014, Stollenga et al 2013] to guide the planner's search.

The planning approaches covered so far were developed to reduce computation time and improve the success rates of feasible path planners. Many of them could be extended for use with asymptotically optimal path planners, but there has been little work investigating the effects of using workspace redundancy with asymptotically optimal planners.

Dragan et al [2011a] modified the Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [Zucker et al 2013] trajectory optimizer to be able to handle a set of goal configurations. They found that considering a set of goals improved the paths that were found by CHOMP. These results suggest that specifying a goal as a set of configurations may allow asymptotically optimal planners, e.g. RRTConnect* [Akgun and Stilman 2011, Jordan and Perez 2013, Klemm et al 2015], to find better paths, which this chapter investigates.

Dragan et al [2011b] attempt to predict goal configurations that would lead to high quality paths being found with CHOMP using a range of machine learning algorithms. Given a set of goal configurations, they manage to select one that allows the planner

to find solutions that are on average 8% worse than those found using the entire goal set. In this case it is better to use CHOMP with the entire goal set because the set of goal configurations has already been computed.

7.3 CONFIGURATION SPACE REDUNDANCY

Some robot arms, such as the UR5, have joints that can perform more than one full rotation. Given one configuration we can construct others that put all of the robots links in exactly the same positions by rotating any of the joints by any number of full revolutions. We will call these *equivalent configurations*, because they put each part of the robot in the same position but are distinct in the robot's configuration space. The configuration space of a two degree of freedom robot with equivalent configurations is shown in Fig. 7.2.

Equivalent configurations are always separated by multiples 2π in each dimension that represents a rotational joint in the robot's configuration space. They can be calculated by adding or subtracting multiples of 2π from positions of rotational joints in the robot's configuration space, while remaining within the joint limits.

The number of equivalent configurations a robot arm has depends on the number of rotations, n, each of its N joints can make:

$$n_{\text{equivalent}} = \prod_{n=1}^{N} n_i \tag{7.1}$$

A robot arm that can make two full rotations with each of its two joints will have four equivalent configurations as shown in Fig. 7.2. A robot arm with joints that can each only make one one full rotation will only have one equivalent configuration. This can also be seen in Fig. 7.2 where there is only one configuration in each 2π by 2π block.

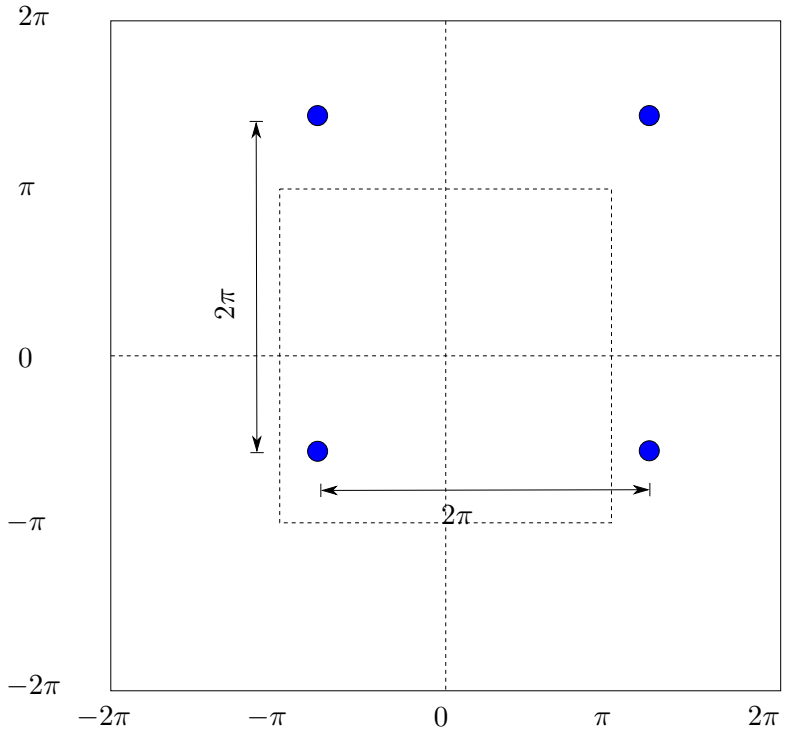


Figure 7.2: The blue dots show equivalent configurations shown in blue for a robot arm with two joints that can each operate in the range $[-2\pi, 2\pi)$. These equivalent configurations are distinct from each other in the robot's configuration space, but put all of the robot's links into the same positions in the workspace.

7.4 FINDING SHORTER PATHS USING CONFIGURATION SPACE AND WORKSPACE REDUNDANCY

The robot's workspace and configuration space redundancy are used to compute a set of goal configurations as shown in Fig. 7.3, that are then used to represent the planner's goal. An inverse kinematic solver is used to generate a predefined number of configurations that put the robot's end-effector in a position to satisfy the workspace goal and also put the robot arm in distinct poses. These configurations represent the robot's workspace redundancy with respect to the task. The equivalent configurations for each of these inverse kinematics solutions is then computed. The resulting configurations represent the robot's workspace and configuration space redundancy with respect to the task. These configurations are then used to represent the path planner's goal.

```
1: function ComputeGoalConfigurations(task_goal)
2:     goal_configs ← {}
3:     ws_goal_configs ← IK solutions that satisfy task
4:     for each g ∈ ws_goal_configs do
5:         equiv_configs ← Equivalent configurations of g
6:         Append equiv_configs to goal_configs
7:     end for
8:     return goal_configs
9: end function
```

Figure 7.3: Algorithm for computing goal configurations for a robot using its workspace and configuration space redundancy.

7.5 EXPERIMENTS

The impact using multiple goals has on the performance of RRTConnect* with integrated short-cutting planner [Paulin et al 2016] configured to minimise the Euclidean path length was tested. Tests were performed on the vine pruning and cubicle picking robot experiments. For both experiments the Euclidean length (sum of Euclidean lengths of each path segment, in radians) and execution time (how long it would take the robot arm to follow the path once the accelerations were assigned in post processing) were recorded. The planner's performance when it was used with a different numbers of the closest goal configurations, and when it was used with different numbers of random goal configurations were tested.

7.6 RESULTS

Two experiments were performed, one on the vine pruning robot and one on the cubicle picking robot. The computation time (how long the planner spent planning), the execution time (how long it would take the robot arm to follow the path), and the cycle times (computation time + execution time) were recorded. In the first experiment the planner's performance with different numbers of randomly-chosen goal configurations for each target (cubicle or cut) was tested. Each of these goal configurations puts the robot arm in a position to achieve its task (cutting a vine or reaching into a cubicle). This is to simulate the case where a randomised IK solver is being used and the user does not know what the closest goal configurations are, these results are shown in Figs. 7.4, 7.8. In the second experiment the planner's performance with different numbers of the closest goal configurations was tested. This is to determine the influence that the proximity of the goal configuration to the start configuration has, these results are shown in Figs. 7.6, 7.10. Planning was performed for 303 cuts in the vine pruning robot experiments, and for reaching into 100 cubicles in the cubicle picking experiments. The same planner parameters were used as in Chapt. 6.

The ranking for the goal configuration used in the shortest path was stored dur-

ing planning. A ranking of n means that the n^{th} closest goal configuration to the start is being used in the shortest path. The goal configuration rankings for the vine pruning and cubicle picking experiments are shown in Figs. 7.5, 7.7, 7.9, 7.11. These figures show how the final configuration ranking changes during planning, and that the shortest paths do not always end at the goal configuration that is closest to the start configuration.

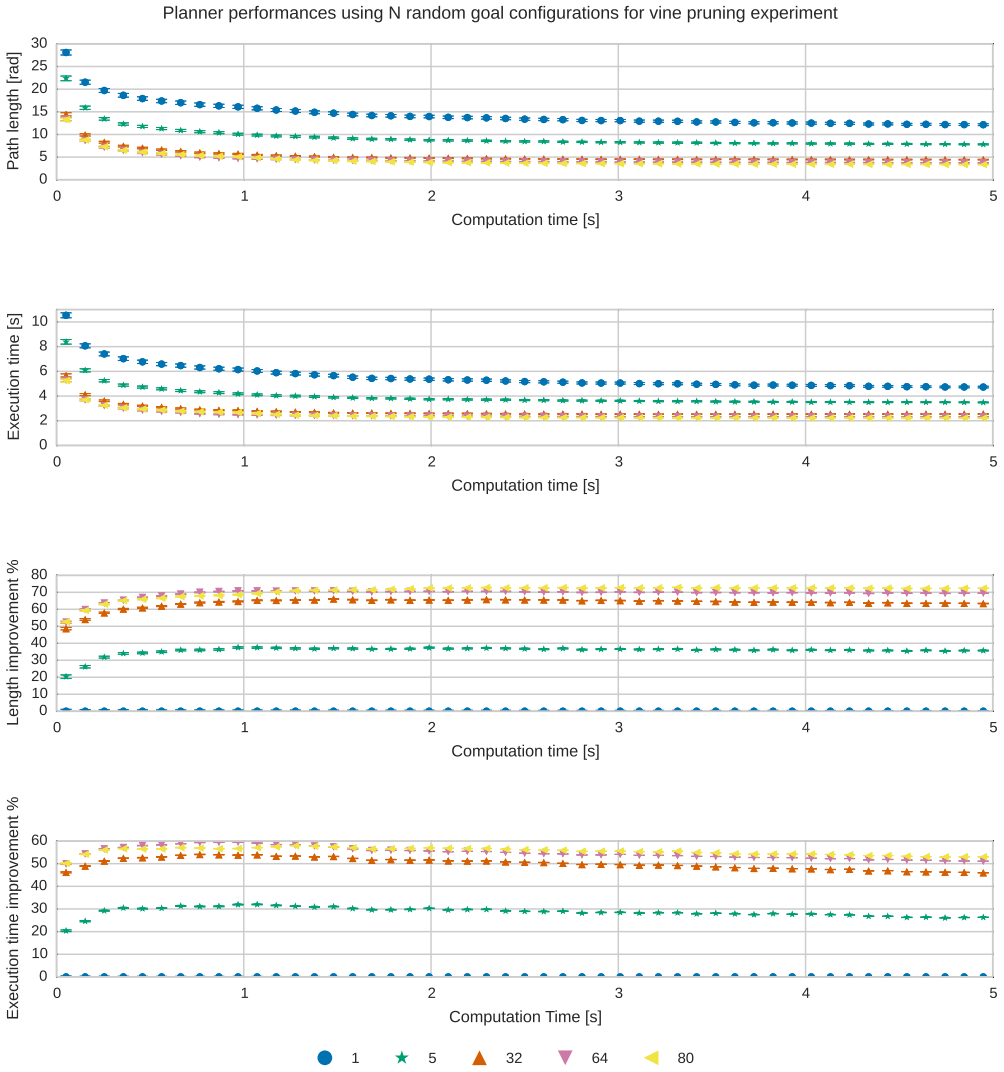


Figure 7.4: Path length and execution times using different numbers of random goals and the improvement over using one goal for the vine pruning experiment. Error bars show the 95% confidence interval.

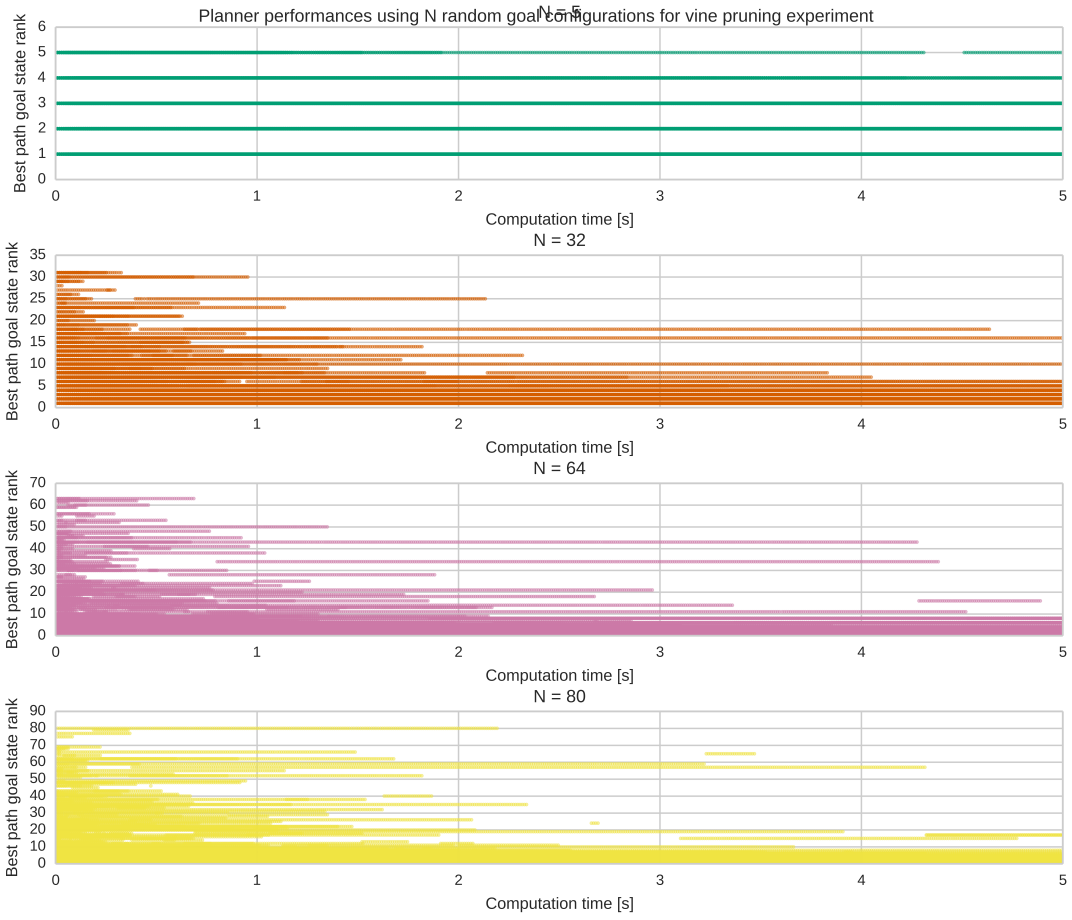


Figure 7.5: Ranking of goal configuration used in shortest path over time for vine pruning experiment. This figure shows that the shortest path found by the planner is frequently not to the closest in configuration space.

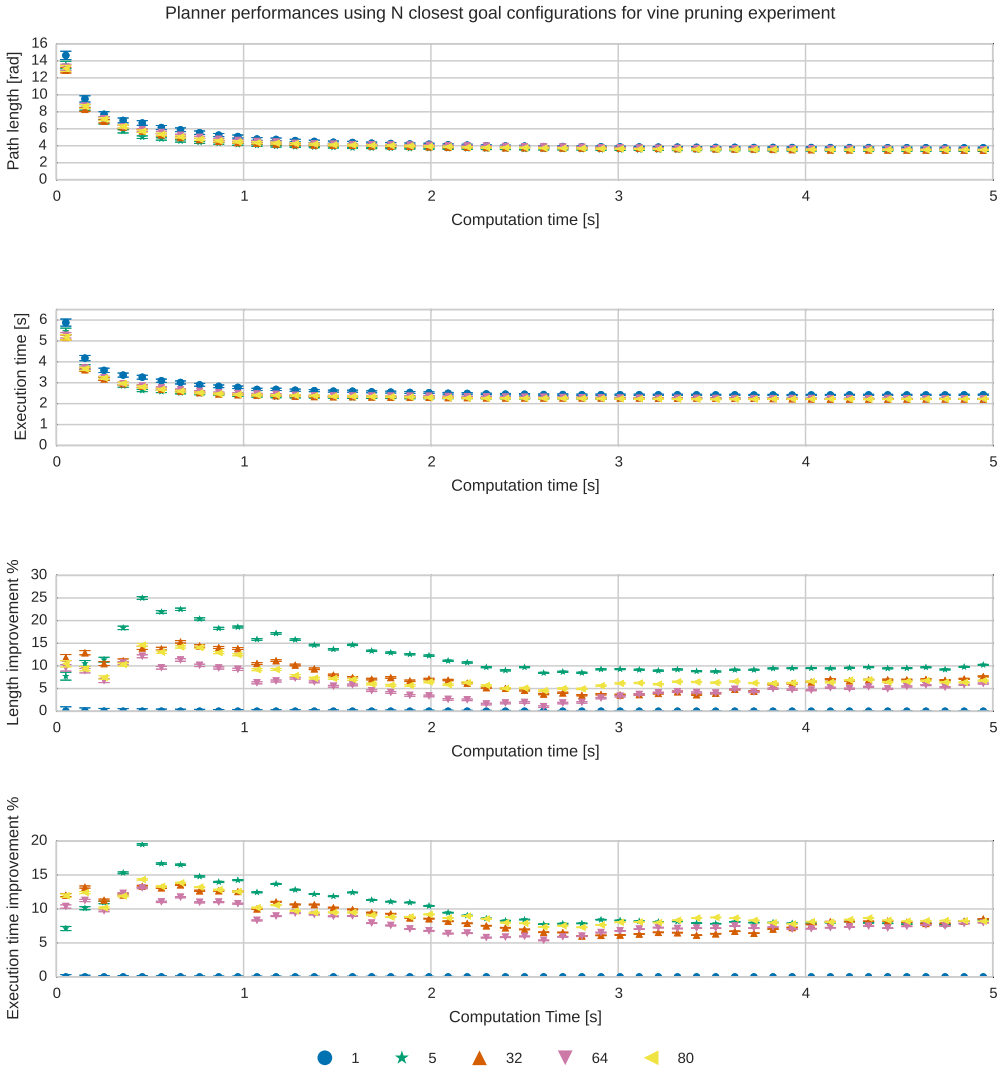


Figure 7.6: Path length and execution times using different numbers of the closest goal configurations and the improvement over using one goal for the vine pruning experiment. Error bars show the 95% confidence interval.

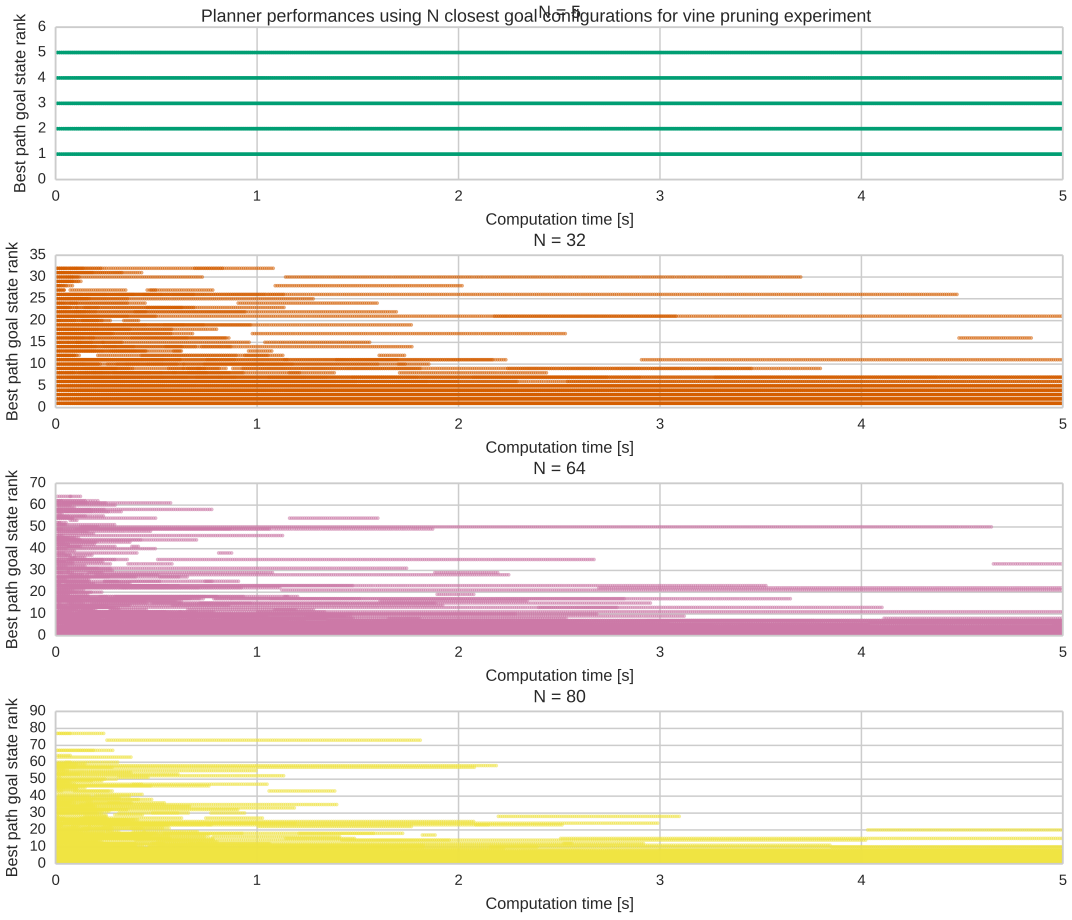


Figure 7.7: Ranking of goal configuration used in shortest path over time for vine pruning experiment. This figure shows that the shortest path found by the planner is frequently not to the closest in configuration space.

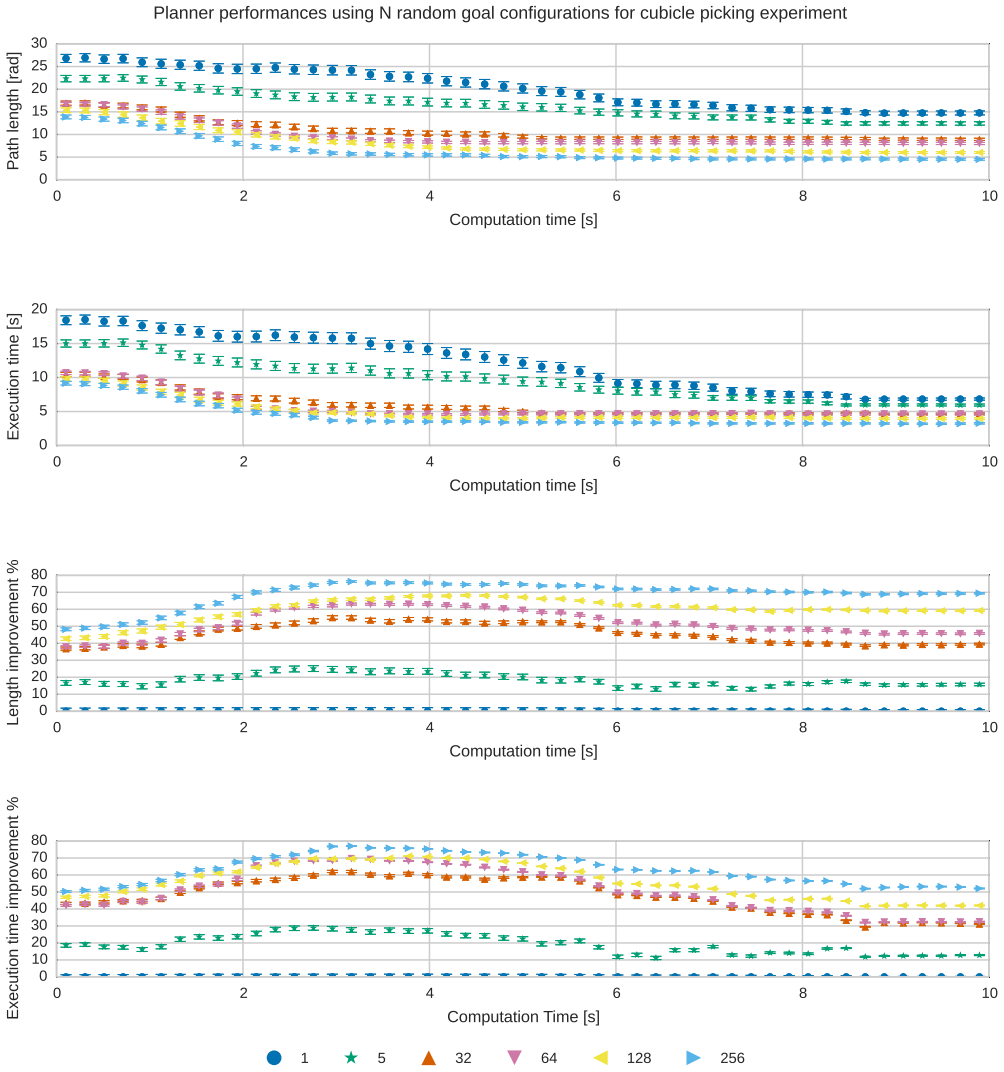


Figure 7.8: Path length and execution times using different numbers of random goals and the improvement over using one goal for the cubicles experiment. Error bars show the 95% confidence interval.

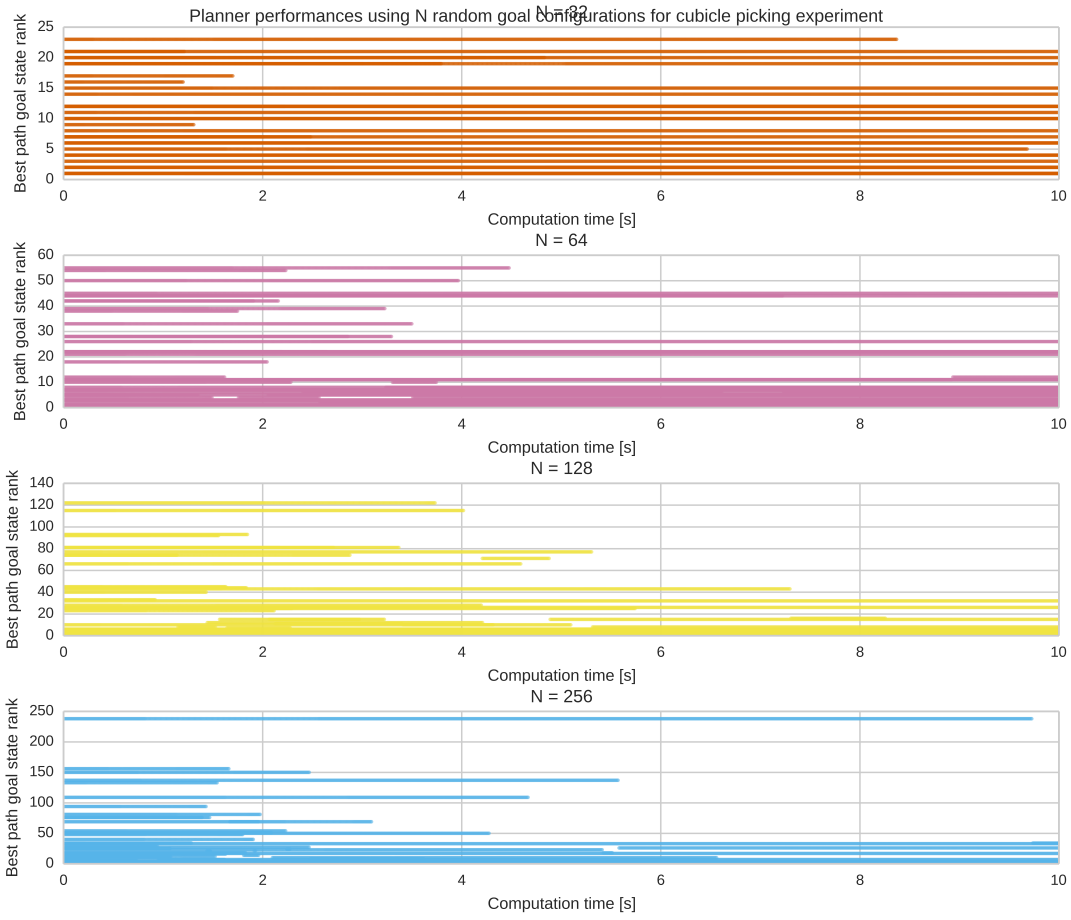


Figure 7.9: Ranking of goal configuration used in shortest path over time for cubicles experiment. This figure shows that the shortest path found by the planner is frequently not to the closest in configuration space.

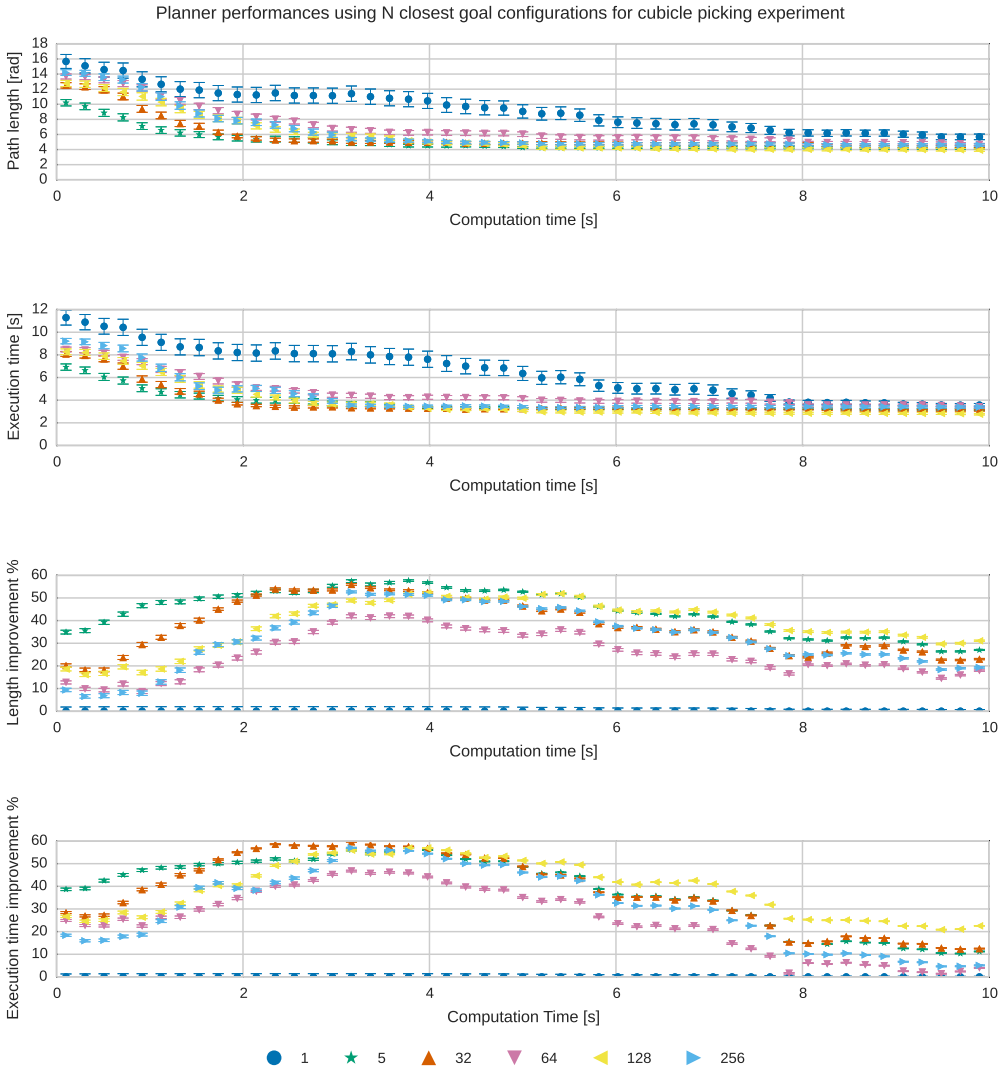


Figure 7.10: Path length and execution times using different numbers of closest goal configurations and the improvement over using one goal for the cubicles experiment. Error bars show the 95% confidence interval.

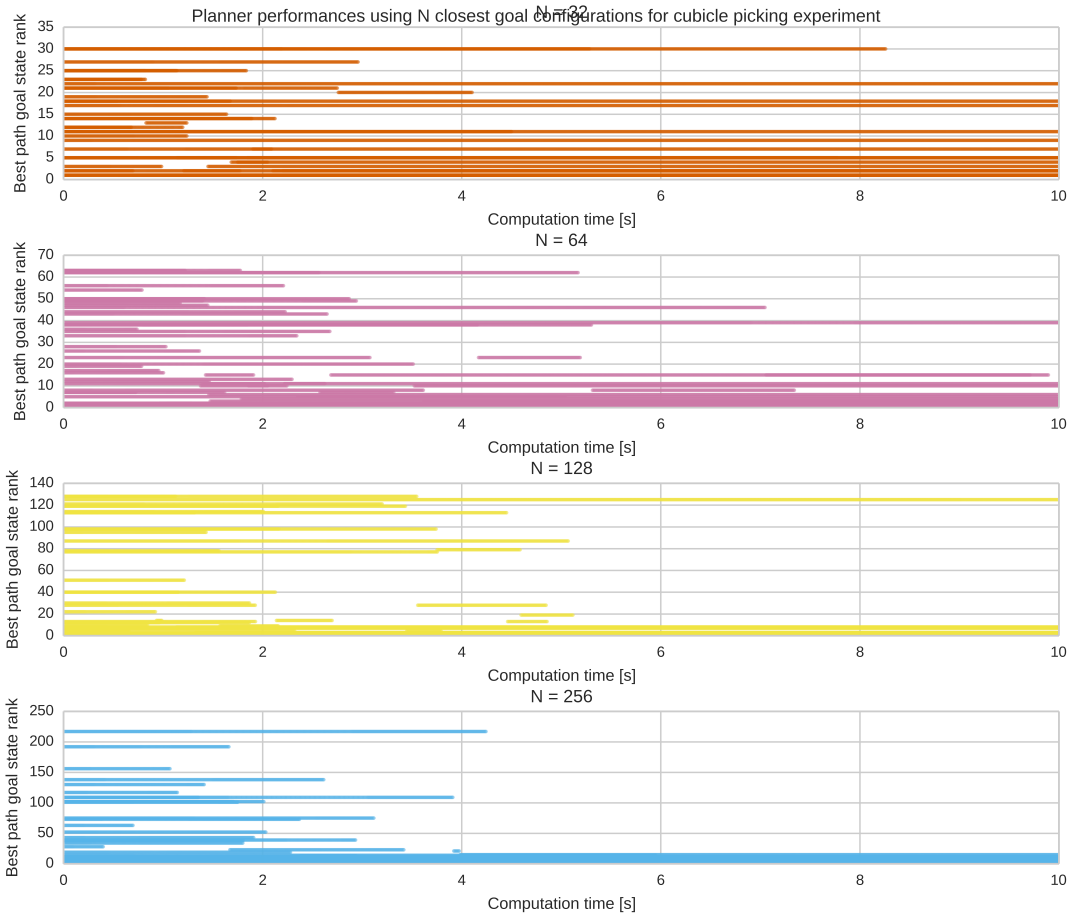


Figure 7.11: Ranking of goal configuration used in shortest path over time for cubicles experiment. This figure shows that the shortest path found by the planner is frequently not to the closest in configuration space.

7.7 DISCUSSION

In our experiments using more than one goal configuration resulted in the planner finding shorter paths that resulted in trajectories which were quicker to execute, even when the first goal configuration was the closest to the start. In most cases using five goal configurations was enough to provide a significant improvement. Using more than 32 goal configurations often did not result in further performance increases. Using multiple goal configurations allows the planner to find paths to alternate goals that might be less obstructed by obstacles.

The results are consistent with Dragan et al [2011a] who found that using goal sets containing 27 configurations improved the cost of paths found by the CHOMP trajectory optimiser by 43% on average. In our experiments we found that using 32 goal configurations resulted in the planner finding paths that were 65% (Fig. 7.4), 15% (Fig. 7.6), 55% (Fig. 7.8) and 55% (Fig. 7.10) shorter than those obtained using only one goal configuration at various stages of planning.

Figures 7.6 and 7.10 show that the planner managed to find shorter paths using a goal set compared to using only the closest goal configuration to the start. This means that sets of goal configurations should be used, even when the closest goal configuration to the start is known.

The planner often finished with solutions that used one of the closest 20 configurations to the start in the vine pruning experiments (Figs. 7.5, 7.7), and one of the 50 closest configurations in the cubicle picking experiments (Figs. 7.9, 7.11). This is consistent with the lack of performance improvements we saw by using more than 32 goal configurations in both experiments. It could be that lower (higher number) ranked goal configurations were too far away to be useful even if they were relatively unobstructed by obstacles.

Using a set of goal configurations may have worked well because it was likely to contain a ‘good’ goal configuration for the query. Alternatively, it may have worked well because the extra goal configurations may have allowed the planner to quickly find

high-quality intermediate solutions, reducing the region of configuration space that was sampled through its use of the informed heuristic [Gammell et al 2014].

It is possible that good performance could be achieved using one high quality goal configuration. Some computation time could be saved if only one goal configuration had to be found using an inverse kinematics routine. This could be achieved if an inverse kinematic solver biased toward solutions in particular regions of configuration space using an appropriate heuristic, although it is unclear what heuristic should be used.

The experiments show that using a set of goal configurations resulted in shorter paths than using the closest goal configuration to the start (Figs. 7.6 and 7.10). This means that a heuristic used to guide the inverse kinematics solver should not only consider the proximity of the goal to the start.

7.8 SUMMARY

Using multiple goal configurations allowed the RRTConnect* path planner with an integrated short-cutting local optimiser to find paths that were shorter and that resulted in trajectories that were faster to execute. These extra goals meant that the planner was able to find paths to different goal configurations that may be shorter compared to a planner that chooses one goal configuration randomly. In a grape vine pruning robot arm experiment the proposed planner reduced execution times by 58%.

The results from this chapter suggest that using a robot arm's redundancy can lead to better solutions being found by a path planner. In particular, it was found that the shortest path found by a planner did not always finish at the goal configuration closest to the start configuration. These results suggest that approaches that have been used for exploiting a robot arm's redundancy with feasible planners, e.g. workspace goal regions [Berenson and Ferguson 2009], may also allow an asymptotically optimal planner to find higher quality paths.

Chapter 8

CONCLUSIONS AND FUTURE WORK

8.1 CONCLUSION

Many robots operating in unpredictable environments require an online path planning algorithm that can quickly compute high quality paths. This path planning algorithm should be able to quickly find high-quality paths in order to allow the robot to operate efficiently. The purpose of the research in this thesis was to investigate approaches to path planning that would quickly provide fast to execute paths for a vine pruning robot. With the exception of Chapt. 7, the research presented in this thesis is applicable to non-redundant robot arms.

In this thesis approaches have been proposed for reducing computation time required by path planning algorithms, and improving the quality of the solutions that they found. The main results of this thesis were a specialised collision detector that allows the robot to quickly compute collision-free paths, the integration of a short-cut local optimiser with the asymptotically optimal RRTConnect* planner that allowed the robot to find fast-to-execute paths on a limited time budget, and an approach to exploit the vine pruning robot arm's redundancy to enable the planner to find faster-to-execute paths. These algorithms were tested on a grape vine pruning robot, and enabled it to efficiently find short paths so that it could quickly prune grape vines.

The specialised collision detector exploits the structure of grape vines to enable fast collision detection. It was 50 times faster than the popular Flexible Collision Library for classifying the collision status of randomly sampled robot arm configurations.

Consequently, using the proposed specialised collision detector resulted in a 28 times speed up in path planning when compared to using the Flexible Collision Library.

Integrating a short-cut local optimiser with the RRTConnect* path planner allowed the vine pruning robot to find fast-to-execute paths on a limited time budget. This approach found paths that were 31% shorter after three seconds of computation time than those found by RRTConnect* without an integrated short-cut local optimiser.

The workspace and configuration space redundancy of the UR5 robot arm was exploited to allow the RRTConnect* with an integrated short-cut optimiser planner to find shorter paths. Exploiting this redundancy resulted in the planner finding paths that were 58% faster to execute than those found without using the robot arm's redundancy.

8.2 FUTURE WORK

The vine pruning robot currently does not consider the order in which cuts are made on the vine. One way to improve the performance of the robot would be to use a tour path planner, e.g. like the one described by Saha et al [2006], to optimise the order that the cuts are made.

The vine pruning robot uses an off-the-shelf six degree of freedom robot arm. This robot arm may not be the most effective robot arm for the task of vine pruning. One way to improve the efficiency of the vine pruning robot would be to identify other robot arms, or possibly custom robot arm designs, that could also perform the task of vine pruning. One option could be using multiple low degree of freedom arms, with one arm located on each side of the plant.

The vine pruning robot uses a spinning router that needs to be 'swiped' through vines to make cuts. Using this method of cutting vines introduces complexity to the pruning problem compared to using conventional secateurs. Collision free paths for the router to swipe through vines at specific cut-points need to be found. This is difficult because the swipe motions need to be 10cm and occur near the head of the plant, which

is the most cluttered area and also where the 3D reconstruction is least accurate. These swipe motions would not need to be computed if secateurs were used instead of the router for cutting the vines.

The vine pruning robot currently needs to be stopped in front of a plant to perform pruning, however, the 3D reconstruction is performed while the robot is moving. The efficiency of the robot could be improved if the pruning were performed while the robot was moving. This was the original goal of the project, but it became very complex when using the six degree of freedom UR5 and when there were errors in the 3D reconstruction. Pruning while moving using the six-degree of freedom UR5 is complex because as the robot moves, possibly unpredictably as the robot moves over uneven ground, the swipe motions used to cut the vines need to be recomputed. The configuration space paths of these swipe motions could change significantly as the robot moves. The problem of pruning while the robot is moving could be significantly easier if (multiple) low degree of freedom robot arms with secateurs are used instead of a six degree of freedom UR5.

REFERENCES

Ademovic, A. and Lacevic, B. (2014), 'Path planning for robotic manipulators via bubbles of free configuration space: Evolutionary approach', *Mediterranean Conference on Control and Automation*, pp. 1323–1328.

Ademovic, A. and Lacevic, B. (2016), 'Path Planning for Robotic Manipulators Using Expanded Bubbles of', In *International Conference on Robotics and Automation*.

Akgun, B. and Stilman, M. (2011), 'Sampling heuristics for optimal motion planning in high dimensions', *International Conference on Intelligent Robots and Systems*.

Alterovitz, R., Patil, S. and Derbakova, A. (2011), 'Rapidly-Exploring Roadmaps : Weighing Exploration vs . Refinement in Optimal Motion Planning', In *International Conference on Robotics and Automation*.

Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C. and Vallejo, D. (1998), 'OBPRM: An Obstacle-Based PRM for 3D Workspaces', *Workshop on the Algorithmic Foundations of Robotics on Robotics*, pp. 155–168.

Arslan, O. (2013), 'Use of Relaxation Methods in Sampling-based Algorithms for Optimal Motion Planning', In *International Conference on Robotics and Automation*, pp. 2421–2428.

Bac, W., van Henten, E., Hemming, J. and Edan, Y. (2014), 'Harvesting Robots for High-value Crops : State-of-the-art Review and Challenges Ahead', *Journal of Field Robotics*, Vol. 31, No. 6.

Barraquand, J., Langlois, B. and Latombe, J.c. (1992), 'Numerical Potential Field Techniques for Robot Path Planning', *Transactions on Systems, Man, and Cybernetics*, Vol. 22, No. 2, pp. 224–241.

Beeson, P. and Ames, B. (2015), 'TRAC-IK: An open-source library for improved solving of generic inverse kinematics', In *International Conference on Humanoid Robots*, pp. 928–935.

Berchtold, S. and Glavina, B. (1994), 'A scalable optimizer for automatically generated manipulator motions', *International Conference on Intelligent Robots and Systems*.

Berenson, D. and Ferguson, D. (2009), 'Manipulation Planning with Workspace Goal Regions', In *International Conference on Robotics and Automation*, pp. 618–624.

Berenson, D., Srinivasa, S. and Kuffner, J. (2011), 'Task Space Regions: A framework for pose-constrained manipulation planning', *International Journal of Robotics Research*, Vol. 30, No. 12, pp. 1435–1460.

Berenson, D., Abbeel, P. and Goldberg, K. (2012), 'A robot path planning framework that learns from experience', *International Conference on Robotics and Automation*, may, pp. 3671–3678.

Bergen, G.V.D. (1997), 'Efficient Collision Detection of Complex Deformable Models using AABB Trees', *Journal of Graphics Tools*, Vol. 2, No. 4, pp. 1–13.

Bertram, D., Kuffner, J., Dillmann, R. and Asfour, T. (2006), 'An integrated approach to inverse kinematics and path planning for redundant manipulators', *International Conference on Robotics and Automation*, pp. 1874–1879.

Bialkowski, J., Otte, M., Karaman, S. and Frazzoli, E. (2016), 'Efficient collision checking in sampling-based motion planning via safety certificates', *International Journal of Robotics Research*, Vol. 35, No. 7, pp. 767 – 796.

Bohlin, R. and Kavraki, E.E. (2000), 'Path planning using lazy PRM', In *International Conference on Robotics and Automation*.

Bohlin, R. and Kavraki, L.E. (2001), 'A Randomized Approach to Robot Path Planning Based on Lazy Evaluation', *Handbook on Randomized Computing*.

Boor, V., Overmars, M.H. and Stappen, a.V.D. (1999), 'The Gaussian sampling strategy for probabilistic roadmap planners'.

Bopardikar, S.D., Englot, B., Speranzon, A. and van den Berg, J. (2016), 'Robust belief space planning under intermittent sensing via a maximum eigenvalue-based bound', *The International Journal of Robotics Research*.

Boser, E., Vapnik, N., Guyon, I.M. and Laboratories, T.B. (1992), 'Training Algorithm Margin for Optimal Classifiers', In *Workshop on Computational Learning Theory*.

Botterill, T., Green, R. and Mills, S. (2013), 'Finding a vine's structure by bottom-up parsing of cane edges', *International Conference on Image and Vision Computing New Zealand*, nov, pp. 112–117.

Botterill, T., Paulin, S., Green, R., Williams, S., Lin, J., Saxton, V., Mills, S., Chen, X. and Corbett-Davies, S. (2016), 'A robot system for pruning grape vines', *The Journal of Field Robotics*.

Brin, S. (1995), 'Near Neighbor Search in Large Metric Spaces', *VLDB Conference. Zurich, Switzerland*, pp. 574–584.

Cai Jianrong, Wang Feng, LüQiang, W.J. (2009), 'Real-time path planning for citrus picking robot based on SBL-PRM', *Transactions of the CSAE*, pp. 158–162.

Chakravorty, S. and Kumar, S. (2011), 'Generalized Sampling-Based Motion Planners', *Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 41, No. 3, pp. 855–866.

Chatzimichali, A.P., Georgilas, I.P. and Tourassis, V.D. (2009), 'Design of an advanced prototype robot for white asparagus harvesting', *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, jul, pp. 887–892.

Chitta, S., Sucan, I. and Cousins, S. (2012), 'Moveit!', *Robotics and Automation Magazine*.

Choudhury, S., Dellin, C. and Srinivasa, S. (2016a), 'Pareto-Optimal Search over Configuration Space Beliefs for Anytime Motion Planning', In *International Conference on Intelligent Robots and Systems*.

Choudhury, S., Gammell, J.D., Barfoot, T.D., Srinivasa, S.S. and Scherer, S. (2016b), 'Regionally Accelerated Batch Informed Trees (RABIT*): A Framework to Integrate Local Information into Optimal Path Planning', In *International Conference on Robotics and Automation*.

Christensen, L. (2000), *Raisin Production Manual*, UCANR Publications.

Cohen, J.D., Lin, M.C., Manocha, D. and Ponamgi, M. (1995), 'I-COLLIDE: An interactive and exact collision detection system for large-scale environments', In *Symposium on Interactive 3D Graphics*.

Coleman, D., Suscan, I., Moll, M., Okada, K. and Correll, N. (2015), 'Experience-Based Planning with Sparse Roadmap Spanners', In *International Conference on Robotics and Automation*, pp. 2–7.

Corbett-davies, S., Botterill, T., Green, R. and Saxton, V. (2012), 'An expert system for automatically pruning vines', In *IVCNZ*.

Correll, N., Bekris, K.E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J.M. and Wurman, P.R. (2016), 'Lessons from the Amazon Picking Challenge', *Arxiv*, Vol. 6, No. 1, pp. 1–14.

Dalibard, S., Nakhaei, A., Lamiraux, F. and Laumond, J.p. (2009), 'Whole-Body Task Planning for a Humanoid Robot : a Way to Integrate Collision Avoidance', In *International Conference on Humanoid Robots*, pp. 355–360.

David Baraff (1992), *Dynamic Simulation of non-penetrating rigid bodies*, PhD thesis, Cornell.

Davidson, J.R., Silwal, A., Hohimer, C.J., Karkee, M., Mo, C. and Zhang, Q. (2016), 'Proof-of-Concept of a Robotic Apple Harvester', In *International Conference on Intelligent Robots and Systems*, pp. 3–8.

de Gier, M.R. (2009), *Control of a robotic arm - Application to on-surface 3D-printing*, PhD thesis.

Devaurs, D., Simeon, T. and Cortes, J. (2016), 'Optimal Path Planning in Complex Cost Spaces with Sampling-Based Algorithms', *Transactions on Automation Science and Engineering*, Vol. 13, No. 2, pp. 415–424.

Dobson, A. and Bekris, K.E. (2014), 'Sparse roadmap spanners for asymptotically near-optimal motion planning', *The International Journal of Robotics Research*, Vol. 33, No. 1.

Dragan, A.D., Ratliff, N.D. and Srinivasa, S.S. (2011a), 'Manipulation planning with goal sets using constrained trajectory optimization', In *International Conference on Robotics and Automation*, may, pp. 4582–4588.

Dragan, A., Gordon, G. and Srinivasa, S. (2011b), 'Learning from Experience in Manipulation Planning: Setting the Right Goals', In *International Symposium on Robotics Research*, jul.

Drumwright, E. and Ng-thow hing, V. (2006), 'Toward Interactive Reaching in Static Environments for Humanoid Robots', In *International Conference on Intelligent Robots and Systems*, pp. 846–851.

Dryden, G. (2014), *2014 viticulture monitoring report*, Technical Report, Ministry for Primary Industries and New Zealand Winegrowers.

Edan, Y. (1995), 'Design of an autonomous agricultural robot', *Applied Intelligence*, Vol. 5, No. 1, jan, pp. 41–50.

Edan, Y., Rogozin, D., Flash, T. and Miles, G.E. (2000), 'Robotic melon harvesting', *IEEE Transactions on Robotics and Automation*, Vol. 16, No. 6, pp. 831–835.

Ellekilde, L.P. and Petersen, H.G. (2013), 'Motion planning efficient trajectories for industrial bin-picking', *The International Journal of Robotics Research*, Vol. 32, No. 9-10, pp. 991–1004.

Ericson, C. (2004), *Real time collision detection*, Elsevier.

Fahimi, F. (2009), *Autonomous Robots*, Springer US, Boston, MA.

FAO (2017), 'FAOSTAT'.

Gammell, J.D., Srinivasa, S.S. and Barfoot, T.D. (2014), 'Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic', In *International Conference on Intelligent Robots and Systems*.

Gammell, J.D., Srinivasa, S.S. and Barfoot, T.D. (2015), 'BIT*: Batch Informed Trees for Optimal Sampling-based Planning via Dynamic Programming on Implicit Random Geometric Graphs', In *International Conference on Robotics and Automation*.

Gasparri, A., Oliva, G. and Panzieri, S. (2009), 'Path planning using a lazy spatial network PRM', *Mediterranean Conference on Control and Automation*, jun, pp. 940–945.

Geraerts, R. (2006), *Sampling-based Motion Planning: Analysis and Path Quality*, PhD thesis.

Geraerts, R., Overmars, M.H. and Overmars, M.H. (2007), 'Creating High-quality Paths for Motion Planning', *The International Journal of Robotics Research*.

Gipson, B., Moll, M. and Kavraki, L.E. (2013), 'Resolution Independent Density Estimation for motion planning in high-dimensional spaces', *International Conference on Robotics and Automation*, may, pp. 2437–2443.

Gottschalk, S., Lin, M.C., Manocha, D. and Hill, C. (1996), 'OBB Tree: A Hierarchical Structure for Rapid Interference Detection', In *Annual Conference on Computer Graphics and Interactive Techniques*, ACM.

Hart, P.E. and Nils, J. (1968), 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths', *Transactions on Systems Science and Cybernetics*, No. 2, pp. 100–107.

Hauser, K. (2015), 'Lazy collision checking in asymptotically-optimal motion planning', *International Conference on Robotics and Automation*, Vol. 2015-June, No. June, pp. 2951–2957.

Hauser, K. and Ng-thow hing, V. (2010), 'Fast Smoothing of Manipulator Trajectories using Optimal Bounded-Acceleration Shortcuts', In *International Conference on Robotics and Automation*.

Hawkins, K.P. (2013), 'Analytic Inverse Kinematics for the Universal Robots UR5/UR10 Arms'.

Hayashi, S., Shigematsu, K., Yamamoto, S., Kobayashi, K., Kohno, Y., Kamata, J. and Kurita, M. (2010), 'Evaluation of a strawberry-harvesting robot in a field test', *Biosystems Engineering*, Vol. 105, No. 2, feb, pp. 160–171.

Hirano, Y., Kitahama, K.i. and Yoshizawa, S. (2005), 'Image-based Object Recognition and Dexterous Hand / Arm Motion Planning Using RRTs for Grasping in Cluttered Scene', In *International Conference on Intelligent Robots and Systems*, pp. 2–7.

Hsu, D., Latombe, J.C. and Motwani, R. (1999), 'Path planning in expansive configuration spaces', *International Journal of Computational Geometry and Applications*.

Hsu, D., Jiang, T., Reif, J. and Sun, Z. (2003), 'The Bridge Test for Sampling Narrow Passages with', In *International Conference on Robotics and Automation*, pp. 1–7.

Hwang, Y.K., Ahuja, N. and Member, S. (1992), 'A Potential Field Approach to Path Planning', *Transactions on Robotics and Automation*, Vol. 8, No. 1, pp. 23–32.

Jaillet, L., Cortés, J. and Siméon, T. (2010), 'Sampling-based path planning on configuration-space costmaps', *Transactions on Robotics*.

Janson, L., Schmerling, E., Clark, A. and Pavone, M. (2015), 'Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions', *The International Journal of Robotics Research*.

Jordan, M. and Perez, A. (2013), *Optimal Bidirectional Rapidly-Exploring Random Trees Random Trees*, Technical Report, MIT.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P. and Schaal, S. (2011), 'STOMP: Stochastic Trajectory Optimization for Motion Planning', In *International Conference on Robotics and Automation*.

Karaman, S. and Frazzoli, E. (2011), 'Sampling-based algorithms for optimal motion planning', *The International Journal of Robotics Research*, jun.

Kavraki, L.E., Svestka, P., Latombe, J.C. and Overmars, M.H. (1996), 'Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces', *Transactions on Robotics and Automation*.

Keselman, L., Verriest, E. and Vela, P.A. (2014), 'Forage RRT - An Efficient Approach To Task-Space Goal Planning for High Dimensional Systems', In *International Conference on Robotics and Automation*, pp. 1572–1577.

Khatib, O. (1986), 'Real-Time Obstacle Avoidance for Manipulators and Mobile Robots', *The International Journal of Robotics Research*, Vol. 5, No. 1, mar, pp. 90–98.

Kilby, M. (1999), *Pruning methods affect yield and fruit quality of merlot and sauvignon blanc grapevines*, Technical Report.

Kirkpatrick, S., Gelatt, C.D. and Vecch, M.P. (1983), 'Optimization by Simulated Annealing', *Science*, Vol. 220, No. 4598, pp. 671–680.

Klemm, S., Oberlander, J., Hermann, A., Roennau, A. and Schamm, T. (2015), 'RRT*-Connect : Faster , Asymptotically Optimal Motion Planning', In *International Conference on Robotics and Biomimetrics*.

Klosowski, J., Held, M., Mitchell, J.S.B., N, K.Z. and Sowizral, H. (1998), 'Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPS', *Transations on Visualization and Computer Graphics*.

Koren, Y. and Borenstein, J. (1991), 'Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation', In *International Conference on Robotics and Automation (ICRA)*, pp. 1398–1404.

Kuffner, J.J. and Lavalle, S.M. (2000), 'RRT-Connect : An Efficient Approach to Single-Query Path Planning', In *International Conference on Robotics and Automation*.

Kuntz, A., Bowen, C. and Alterovitz, R. (2016), 'Interleaving Optimization with Sampling-Based Motion Planning (IOS-MP): Combining Local Optimization with Global Exploration', *ArXiv Preprint*.

Lacevic, B. and Rocco, P. (2010), 'Towards a complete safe path planning for robotic manipulators', *International Conference on Intelligent Robots and Systems*, pp. 5366–5371.

Lacevic, B. and Rocco, P. (2013), 'Safety-oriented path planning for articulated robots', *Robotica*, Vol. 31, No. 06, pp. 861–874.

Lacevic, B., Osmankovic, D. and Ademovic, A. (2016), 'Burs of Free C-space : a Novel Structure for Path Planning', In *International Conference on Robotics and Automation*.

Ladd, A.M. and Kavraki, L.E. (2005), 'Motion Planning in the Presence of Drift, Underactuation and Discrete System Changes.', *Robotics: Science and Systems*.

Lauterbach, C., Mo, Q. and Manocha, D. (2010), 'gProximity : Hierarchical GPU-based Operations for Collision and Distance Queries', *Computer Graphics Forum*.

LaValle, S.M. (1998), *Rapidly-Exploring Random Trees: A New Tool for Path Planning*, Technical Report.

LaValle, S.M. (2001), 'Randomized Kinodynamic Planning', *The International Journal of Robotics Research*, may.

Lecun, Y., Bengio, Y. and Hinton, G. (2015), 'Deep learning', *Nature*.

Lee, J.J.H., Frey, K., Fitch, R. and Sukkareith, S. (2014), 'Fast Path Planning for Precision Weeding', In *Australasian Conference on Robotics and Automation*.

Liu, W., Kantor, G., De la Torre, F. and Zheng, N. (2012), 'Image-based tree pruning', *International Conference on Robotics and Biomimetics (ROBIO)*, No. 3, dec, pp. 2072–2077.

Long, P., Liu, W. and Pan, J. (2016), 'Deep-Learned Collision Avoidance Policy for Distributed Multi-Agent Navigation', *ArXiv*.

Lozano-perez, T. (1983), 'Spatial Planning: A Configuration Space Approach', *Transations on Computers*.

Lozano-prez, T. and Wesley, M.A. (1979), 'An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles', *Communications of the ACM*, Vol. 22, No. 10.

Luo, J. and Hauser, K. (2014), 'An Empirical Study of Optimal Motion Planning', In *International Conference on Intelligent Robots and Systems*.

Marquardt and Donald. W. (1963), 'An algorithm for least-squares estimation of nonlinear parameters', *Journal of the Society for Industrial {&} Applied Mathematics*, Vol. 11, No. 2.

Meagher, D. (1982), 'Geometric Modeling Using Octree Encoding', *Computer Graphics and Image Processing*.

Missiuro, P.E. and Roy, N. (2006), 'Adapting probabilistic roadmaps to handle uncertain maps', *International Conference on Robotics and Automation*, Vol. 2006, No. May, pp. 1261–1267.

Moll, M. and Sucan, I. (2015), 'Benchmarking Motion Planning Algorithms', *Robotics {&} Automation Magazine*, No. August.

Nasir, J., Islam, F., Malik, U., Ayaz, Y., Hasan, O., Khan, M. and Muhammad, M.S. (2013), 'RRT*-SMART: A rapid convergence implementation of RRT*', *International Journal of Advanced Robotic Systems*.

Newman, W.S. and Hogan, N. (1987), 'High speed robot control and obstacle avoidance using dynamic potential functions', *International Conference on Robotics and Automation*, pp. 14–24.

Nguyen, T.T., Kayacan, E., Baedemaeker, J.D. and Saeys, W. (2013), 'Task and motion planning for apple harvesting robot', In *IFAC Conference on Modelling and Control in Agriculture, Horticulture and Post Harvest Industry*.

Nir, O. (2014), *Modelling of tree structure using robotic arm for pruning*, PhD thesis.

Noguchi, N. and Terao, H. (1997), 'Path planning of an agricultural mobile robot by neural network and genetic algorithm', *Computers and Electronics in Agriculture*, Vol. 18, No. 2-3, aug, pp. 187–204.

NZ Winegrowers (2016), 'New Zealand Wine Industry Key Performance Indicators: June Year End 2016'.

Otte, M. and Frazzoli, E. (2015), 'RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning', *International Journal of Robotics Research*.

Pan, J. and Manocha, D. (2012), 'GPU-based parallel collision detection for fast motion planning', *The International Journal of Robotics Research*.

Pan, J. and Manocha, D. (2015), 'Efficient Configuration Space Construction and Optimization for Motion Planning', *Engineering*.

Pan, J. and Manocha, D. (2016), 'Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing', *International Journal of Robotics Research*.

Pan, J., Chitta, S. and Manocha, D. (2012), 'FCL: A general purpose library for collision and proximity queries', *International Conference on Robotics and Automation*.

Paulin, S., Botterill, T., Green, R. and Chen, X. (2016), 'Integrating asymptotically-optimal path planning with local optimization', *Arxiv*, pp. 1–14.

Phillips, M., Cohen, B.J., Chitta, S. and Likhachev, M. (2012), 'E-Graphs: Bootstrapping Planning with Experience Graphs', *Robotics Science and Systems*.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. and Ng, A. (2009), 'ROS : an open-source Robot Operating System', *ICRA workshop on open source software*.

Quinlan, S. and Khatib, O. (1993), 'Elastic Bands: Connecting Path Planning and Control', In *International Conference on Robotics and Automation (ICRA)*, pp. 802–807.

Ratliff, N., Zucker, M., Bagnell, J.A. and Srinivasa, S. (2009), 'CHOMP: Gradient optimization techniques for efficient motion planning', *International Conference on Robotics and Automation*.

Saha, M., Sanchez-Ante, G., Roughgarden, T. and Latombe, J.C. (2006), 'Planning Tours of Robotic Arms among Partitioned Goals', *The International Journal of Robotics Research*, Vol. 25, No. 3, pp. 207–223.

Salzman, O. and Halperin, D. (2016), 'Asymptotically Near-Optimal RRT for Fast, High-Quality Motion Planning', *Transactions on Robotics*.

Sánchez, G. and Latombe, J.C. (2003a), 'A single-query bi-directional probabilistic roadmap planner with lazy collision checking', In *International Symposium Robotics Research*, Springer.

Sánchez, G. and Latombe, J.C. (2003b), 'A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking', In Jarvis, R.A. and Zelinsky, A. (Eds.), *Robotics Research: The Tenth International Symposium*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 403–417.

Scarfe, A.J., Flemmer, R.C., Bakker, H.H. and Flemmer, C.L. (2009), 'Development of an autonomous kiwifruit picking robot', *International Conference on Automation, Robotics and Applications*.

Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K. and Abbeel, P. (2014), 'Motion planning with sequential convex optimization and convex collision checking', *The International Journal of Robotics Research*.

Schwartz, J.T. and Sharir, M. (1983), 'On the piano movers problem. II. General techniques for computing topological properties of real algebraic manifolds', *Advances in Applied Mathematics*, Vol. 4, No. 3, sep, pp. 298–351.

Starek, J.A., Schmerling, E., Janson, L. and Pavone, M. (2015), 'An Asymptotically-Optimal Sampling-Based Algorithm for Bi-directional Motion Planning', In *International Conference on Robotics and Automation*.

Stilman, M., Schamburek, J.u. and Kuffner, J. (2007), 'Manipulation Planning Among Movable Obstacles', In *International Conference on Intelligent Robots and Systems*.

Stollenga, M., Pape, L., Frank, M. and Alexander, F. (2013), 'Task-Relevant Roadmaps : A Framework for Humanoid Motion Planning', In *International Conference on Intelligent Robots and Systems*, pp. 5772–5778.

Sucan, I. and Kavraki, L. (2009a), 'Kinodynamic motion planning by interior-exterior cell exploration', *Algorithmic Foundation of Robotics VIII*, pp. 1–16.

Sucan, I.A. and Kavraki, L.E. (2009b), 'On the performance of random linear projections for sampling-based motion planning', *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pp. 2434–2439.

Sucan, I.A. and Kavraki, L.E. (2010), 'On the implementation of single-query sampling-based motion planners', In *International Conference on Robotics and Automation*.

Sucan, I., Moll, M. and Kavraki, E. (2012), 'The open motion planning library', *IEEE Robotics & Automation Magazine*.

Thomas, S., Morales, M., Tang, X. and Amato, N.M. (2007), 'Biasing Samplers to Improve Motion Planning Performance', In *International Conference on Robotics and Automation*.

Vahrenkamp, N. and Asfour, T. (2007), 'Efficient Motion Planning for Humanoid Robots using Lazy Collision Checking and Enlarged Robot Models', In *International Conference on Intelligent Robots and Systems*, pp. 3062–3067.

Valencia, R., Andrade-Cetto, J. and Porta, J.M. (2010), 'Path planning with pose SLAM', In *International Conference on Robotics and Automation*, pp. 1050–1059.

Vande Weghe, M., Ferguson, D. and Srinivasa, S.S. (2007), 'Randomized Path Planning for Redundant Manipulators without Inverse Kinematics', In *International Conference on Humanoid Robots*.

(2017), 'Vision Robotics Corporation.'.

Wilmarth, S., Amato, N.M. and Stiller, P.F. (1999), 'MAPRM: a probabilistic roadmap planner with sampling on the medial axis of the free space', *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, Vol. 2, pp. 1024–1031.

Zhong, C. and Liu, H. (2016), 'A Region-specific Hybrid Sampling Method for Optimal Path Planning', *International Journal of Advanced Robotic Systems*.

Zucker, M., Ratliff, N., Dragan, a.D., Pivtoraiko, M., Klingensmith, M., Dellin, C.M., Bagnell, J.a. and Srinivasa, S.S. (2013), 'CHOMP: Covariant Hamiltonian optimization for motion planning', *The International Journal of Robotics Research*, Vol. 32, No. 9-10, pp. 1164–1193.

APPENDIX A PUBLICATIONS

A comparison of sampling based path planners for a vine pruning robot arm. **S. Paulin**, T. Botterill, J. Lin, X. Chen, R. Green. International Conference on Automation, Robotics and Applications, 2015.

A specialised collision detector for grape vines. **S. Paulin**, T. Botterill, X. Chen, R. Green. Australasian Conference on Robotics and Automation. 2015.

A robot system for pruning grape vines. T. Botterill, **S. Paulin**, R. Green, S. Williams, J. Lin, V. Saxton, S. Mills, X. Chen, S. Corbett-Davies. Journal of Field Robotics. 2016.

Integrating asymptotically-optimal path planning with local optimization. **S. Paulin**, T. Botterill, X. Chen, R. Green. Submitted to Robotics and Autonomous Systems.

Finding shorter paths for robot arms using their redundancy. **S. Paulin**, T. Botterill, X. Chen, R. Green. Submitted to Autonomous Robots.

APPENDIX B NEIGHBOURHOOD DEFINITIONS

Table 1 shows the definitions of neighbourhood sizes in terms of a radius r_n and a number of nearest neighbours k_n. Symbols are defined as follows:

d The dimension of the configuration space.

$\mu(C_{\text{free}})$ The volume of the free part of configuration space. When this is not known the volume of the entire configuration space can be used.

ζ_d The volume of the unit ball in d-dimensional Euclidean space.

η A tuning parameter. In RRT* and RRTConnect* this is called *range*.

Table 1: Neighbourhood definitions for some popular asymptotically optimal sampling based planners. γ and k are intermediate variables, they represent minimum values required for asymptotic optimality. r_n defines the neighbourhood in terms of a radius. k_n defines the neighbourhood in terms of a number of nearest neighbours.

Planner	$\gamma >$	r_n	$k >$	k_n
PRM*	$2(1+1/d)^{\frac{1}{d}}\left(\frac{\mu(C_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}}$	$\gamma(\frac{\log n}{n})^{\frac{1}{d}}$	$e(1+1/d)$	$k\log n$
RRT*	$(2(1+1/d))^{\frac{1}{d}}\left(\frac{\mu(C_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}}$	$\min\left(\frac{\gamma}{\zeta_d}\frac{\log n}{n}, \eta\right)$	$2^{d+1}e(1+1/d)$	$k\log n$
RRTConnect*	$(2(1+1/d))^{\frac{1}{d}}\left(\frac{\mu(X_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}}$	$\min\left(\frac{\gamma}{\zeta_d}\frac{\log n}{n}, \eta\right)$	$2^{d+1}e(1+1/d)$	$k\log n$
FMT*	$(1+\eta)2(1/d)^{\frac{1}{d}}\left(\frac{\mu(C_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}}$	$\gamma\left(\frac{\log n}{n}\right)^{\frac{1}{d}}$	$3^d e(1+1/d)$	$k\log n$
BIT*	$(1+\eta)2(1/d)^{\frac{1}{d}}\left(\frac{\mu(C_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}}$	$\gamma\left(\frac{\log n}{n}\right)^{\frac{1}{d}}$	$3^d e(1+1/d)$	$k\log n$

APPENDIX C UR5 TECHNICAL SPECIFICATIONS

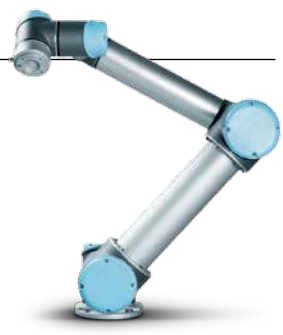
UNIVERSAL ROBOTS

UR5 Technical specifications

Item no. 110105

6-axis robot arm with a working radius of 850 mm / 33.5 in

Weight:	18.4 kg / 40.6 lbs
Payload:	5 kg / 11 lbs
Reach:	850 mm / 33.5 in
Joint ranges:	+/- 360°
Speed:	All joints: 180°/s. Tool: Typical 1 m/s. / 39.4 in/s.
Repeatability:	+/- 0.1 mm / +/- 0.0039 in (4 mils)
Footprint:	Ø149 mm / 5.9 in
Degrees of freedom:	6 rotating joints
Control box size (WxHxD):	475 mm x 423 mm x 268 mm / 18.7 x 16.7 x 10.6 in
I/O ports:	Controlbox / Tool conn. Digital in 16 / 2 Digital out 16 / 2 Analog in 2 / 2 Analog out 2 / -
I/O power supply:	24 V 2A in control box and 12 V/24 V 600 mA in tool
Communication:	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP
Programming:	Polyscope graphical user interface on 12 inch touchscreen with mounting
Noise:	Comparatively noiseless
IP classification:	IP54
ISO Class Cleanroom robot arm:	5
ISO Class Cleanroom control box:	6
Power consumption:	Approx. 200 watts using a typical program
Collaboration operation:	15 Advanced Safety Functions Tested in accordance with: EN ISO 13849:2008 PL d EN ISO 10218-1:2011, Clause 5.4.3
Materials:	Aluminum, PP plastic
Temperature:	The robot can work in a temperature range of 0-50°C
Power supply:	100-240 VAC, 50-60 Hz
Cabling:	Cable between robot and control box (6 m / 236 in) Cable between touchscreen and control box (4.5 m / 177 in)



Universal Robots A/S
Energivej 25
DK-5260 Odense S
Denmark
+45 89 93 89 89

www.universal-robots.com
sales@universal-robots.com

APPENDIX D SAMPLE PLANT DATA

This appendix provides some sample data output by the 3D reconstruction software.

CAMERA TRANSFORMS

Example output for the transformation matrices:


```
<CAMPOSE>
<camPose class_id="0" tracking_level="0" version="0">
<time class_id="1" tracking_level="0" version="0">
<secondsSinceStart>81.333333333333329</secondsSinceStart>
<framesSinceStart>2440</framesSinceStart>
</time>
<locationMean class_id="2" tracking_level="0" version="0">
<X>2.1798972002448438</X>
<Y>-0.0043368050023796737</Y>
<Z>-0.0023462664290978268</Z>
</locationMean>
<locationCovariance class_id="3" tracking_level="0" version="0">
<covMat00>0.00040000000000000002</covMat00>
<covMat01>0</covMat01>
<covMat02>0</covMat02>
<covMat10>0</covMat10>
<covMat11>0.00040000000000000002</covMat11>
<covMat12>0</covMat12>
```

<covMat20>0</covMat20>

<covMat21>0</covMat21>

<covMat22>0.00040000000000000002</covMat22>

</locationCovariance>

<globalOrientation class_id="4" tracking_level="0" version="0">

<qx>0.012794350204725193</qx>

<qy>-0.010304655711501508</qy>

<qz>-0.002899666521239656</qz>

<qw>0.99986084562181532</qw>

</globalOrientation>

<R00>0.99977081200946605</R00>

<R01>0.0055348432920717559</R01>

<R02>-0.02068064224498645</R02>

<R10>-0.0060622087877200441</R10>

<R11>0.99965579307380892</R11>

<R12>-0.025525379499397271</R12>

<R20>0.020532244849187819</R20>

<R21>0.025644899760115438</R21>

<R22>0.9994602373470125</R22>

</camPose>

<velocity>

<X>0.033083250108251888</X>

<Y>-0.00032395361172093121</Y>

<Z>-0.00046633414187689283</Z>

</velocity>

</CAMPOSE>

PLANT

Example output data for a vine with all but one polyline and one sphere for the head model removed:


```
<STRUCTURE>
<polyline class_id="0" tracking_level="0" version="0">
<GUID>9</GUID>
<parentCutGUID>6</parentCutGUID>
<point class_id="1" tracking_level="0" version="0">
<X>0.25706779935016755</X>
<Y>-0.22447819146566914</Y>
<Z>1.1718924486313607</Z>
</point>
<thickness>0.005695061232725 4799</thickness>
<point>
<X>0.28030736419089664</X>
<Y>-0.28255738310325201</Y>
<Z>1.1959032595759729</Z>
</point>
<thickness>0.005695061232725 4799</thickness>
<point>
<X>0.31515542192052637</X>
<Y>-0.43407950017492208</Y>
<Z>1.249618727015084</Z>
</point>
<thickness>0.005695061232725 4799</thickness>
</polyline>
<polyline>
<GUID>10</GUID>
```

<parentCutGUID>6</parentCutGUID>
<point>
<X>0.22743695810273987</X>
<Y>-0.25876811929205784</Y>
<Z>1.1970940353545563</Z>
</point>
<thickness>0.006475998047273677</thickness>
<point>
<X>0.26343947134730827</X>
<Y>-0.33763960327479409</Y>
<Z>1.2240476870701529</Z>
</point>
<thickness>0.006475998047273677</thickness>
<point>
<X>0.287572119291511</X>
<Y>-0.41658288758014106</Y>
<Z>1.2477450753523427</Z>
</point>
<thickness>0.006475998047273677</thickness>
<point>
<X>0.29726854715794065</X>
<Y>-0.44393042549719386</Y>
<Z>1.2592097558694904</Z>
</point>
<thickness>0.006475998047273677</thickness>
<point>
<X>0.35720113249670893</X>
<Y>-0.57955580902530057</Y>
<Z>1.3371424383777444</Z>
</point>

```
<thickness>0.0064759980472733677</thickness>
</polyline>
<headPart class_id="2" tracking_level="0" version="0">
<X>0.56259770853394953</X>
<Y>0.62528707384672499</Y>
<Z>0.95274534329325888</Z>
<W>0.090991237937857058</W>
</headPart>
</STRUCTURE>
```


CUT POINTS

Example output data for a set of cutpoints:


```
<CUTPOINTSET>
<cutset class_id="0" tracking_level="0" version="0">
<plantGUID>116</plantGUID>
<rank>0</rank>
<caneToCut class_id="1" tracking_level="0" version="0">
<caneGUID>139</caneGUID>
<cutType>2</cutType>
<cutPosition>0.13458967641014052</cutPosition>
</caneToCut>
<caneToCut>
<caneGUID>201</caneGUID>
<cutType>2</cutType>
<cutPosition>0.02</cutPosition>
</caneToCut>
<caneToCut>
<caneGUID>144</caneGUID>
```

<cutType>2</cutType>
<cutPosition>0.018675319323439382</cutPosition>
</caneToCut>
<caneToCut>
<caneGUID>151</caneGUID>
<cutType>1</cutType>
<cutPosition>0.09999999999999978</cutPosition>
</caneToCut>
<caneToCut>
<caneGUID>163</caneGUID>
<cutType>1</cutType>
<cutPosition>0.10000000000000001</cutPosition>
</caneToCut>
<caneToCut>
<caneGUID>147</caneGUID>
<cutType>2</cutType>
<cutPosition>0.01755837089427844</cutPosition>
</caneToCut>
</cutset>

</CUTPOINTSET>